

Power Compiler™

User Guide

Version L-2016.03-SP4, September 2016

SYNOPSYS®

Copyright Notice and Proprietary Information

©2016 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>. All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Copyright Notice for the Command-Line Editing Feature

© 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3.All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

- 4.Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the Line-Editing Library

© 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

- 1.The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
- 2.The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
- 3.Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
- 4.This notice may not be removed or altered.

Contents

About This User Guide	xxvi
Customer Support.	xxviii
1. Introduction to Power Compiler	
Power Compiler Methodology.	1-2
Power Library Models.	1-2
Power Analysis Technology	1-3
Power Optimization Technology	1-4
Getting Started With Power Compiler	1-5
Library Requirements	1-5
Command-Line Interface	1-6
Graphical User Interface	1-6
License Requirements	1-7
Reading and Writing Designs	1-8
2. Power Compiler Design Flow	
Power in the Design Cycle	2-2
Power Optimization and Analysis Flow.	2-3
Simulation	2-5
Enable Power Optimization	2-5
Synthesis and Power Optimization	2-5
Power Analysis and Reporting.	2-5

3. Power Modeling and Calculation

Power Types	3-2
Static Power	3-2
Dynamic Power	3-2
Switching Power	3-2
Internal Power	3-3
Calculating Power	3-4
Leakage Power Calculation	3-4
Multithreshold Voltage Libraries	3-5
Internal Power Calculation	3-6
NLDM Models	3-7
State and Path Dependency	3-8
Rise and Fall Power	3-9
Switching Power Calculation	3-10
Dynamic Power Calculation	3-10
Dynamic Power Unit Derivation	3-10
Power Calculation for Multirail Cells	3-12
Using CCS Power Libraries	3-14
Voltage Scaling	3-14
Script Examples for Voltage Scaling	3-15

4. Generating SAIF Files

About Switching Activity	4-2
Introduction to SAIF Files	4-2
Generating SAIF Files	4-3
Generating SAIF Files From Simulation	4-4
Generating SAIF Files From SystemVerilog or Verilog Simulations	4-5
Generating SAIF Files From VHDL Simulation	4-13
Generating SAIF Files From VCD Files	4-14
Converting a VCD File to a SAIF File	4-14
Generating SAIF Files from FSDB Output Files	4-15
Verilog Switching Activity Examples	4-15
RTL Example	4-15
Verilog Design Description	4-15
RTL Testbench	4-17
RTL SAIF File	4-18

Gate-Level Example	4-20
Gate-Level Verilog Module	4-20
Verilog Testbench	4-21
Gate-Level SAIF File	4-22
VHDL Switching Activity Example	4-23
VHDL Design Description	4-23
RTL Testbench.	4-24
RTL SAIF File	4-24
5. Annotating Switching Activity	
Types of Switching Activity to Annotate	5-2
Annotating Switching Activity Using RTL SAIF Files	5-2
Using the Name-Mapping Database	5-3
Integrating the RTL Annotation With PrimeTime PX	5-4
Annotating Switching Activity Using Gate-Level SAIF Files	5-4
Reading SAIF Files Using the read_saif Command	5-5
Reading SAIF Files Using the merge_saif Command	5-6
Annotating Inferred Switching Activity.	5-7
Annotating Switching Activity Using the set_switching_activity Command.	5-8
Fully Versus Partially Annotating the Design	5-11
Analyzing the Switching Activity Annotation	5-12
Removing the Switching Activity Annotation.	5-12
Design Objects Without Annotated Switching Activity	5-13
Default Switching Activity Values.	5-13
Propagating the Switching Activity.	5-14
Deriving the State- and Path-Dependent Switching Activity	5-15
6. Performing Power Analysis	
Overview.	6-2
Identifying Power and Accuracy	6-2
Factors Affecting the Accuracy of Power Analysis	6-3
Switching Activity Annotation	6-3
Delay Model	6-4
Switching Activity Propagation and Accuracy.	6-4

Overriding Library Power Characterization	6-5
Performing Gate-Level Power Analysis	6-5
Using the report_power Command	6-6
Using the report_power_calculation Command	6-9
Analyzing Power With Partially Annotated Designs	6-9
Power Correlation	6-10
Performing Power Correlation	6-11
Power Correlation Script	6-11
Analyzing the Design For Power Analysis	6-12
Characterizing a Design for Power	6-13
Reporting the Power Attributes of Library Cells	6-15
Power Reports	6-15
Power Report Summary	6-17
Net Power Report	6-18
Cell Power Report	6-19
Group Report	6-20
Hierarchical Power Reports	6-21
Power Report for Block Abstraction	6-23
7. Clock Gating	
Introduction to Clock Gating	7-3
Using Clock-Gating Conditions	7-5
Clock-Gating Conditions	7-5
Enable Condition	7-6
Setup Condition	7-8
Enabling or Disabling Clock Gating on Design Objects	7-9
Inserting Clock Gates	7-10
Using the compile_ultra -gate_clock Command	7-10
Using the insert_clock_gating Command	7-10
Clock-Gate Insertion in Multivoltage Designs	7-11
Clock Gating Flows	7-11
Inserting Clock Gates in the RTL Design	7-11
Inserting Clock Gates in Gate-Level Design	7-12

Ensuring Accuracy When Using Ideal Clocks.	7-13
Specifying Clock-Gate Latency.	7-13
The set_clock_latency Command	7-14
The set_clock_gate_latency Command.	7-15
Applying Clock-Gate Latency	7-17
Resetting Clock-Gate Latency	7-17
Comparison of the Clock-Gate Latency Specification Commands	7-17
Calculating the Clock Tree Delay From Clock-Gating Cell to Registers	7-18
Specifying Setup and Hold	7-19
Predicting the Impact of Clock Tree Synthesis	7-21
Choosing a Value for Setup	7-22
Choosing a Value for Hold	7-23
Clock-Gating Styles	7-23
Default Clock-Gating Style.	7-24
Selecting Clock-Gating Style	7-26
Choosing Gating Logic	7-27
Choosing an Integrated Clock-Gating Cell	7-27
Choosing a Configuration for Discrete Gating Logic.	7-29
Choosing a Simple Gating Cell by Name	7-34
Choosing a Simple Gating Cell and Library by Name	7-34
Designating Simple Cells Exclusively for Clock Gating	7-34
Choosing a Specific Latch and Library	7-36
Choosing a Latch-Free Style	7-36
Improving Testability	7-37
Connecting the Test Ports Throughout the Hierarchy.	7-42
Using Instance-Specific Clock-Gating Style	7-43
Modifying the Clock-Gating Structure	7-46
Changing a Clock-Gated Register to Another Clock-Gating Cell	7-46
Removing Clock-Gating Cells From the Design	7-47
Rewiring Clock Gating After Retiming	7-47
Integrated Clock-Gating Cells.	7-48
Integrated Clock-Gating Cell Attributes	7-48
Pin Attributes	7-50
Timing Considerations	7-51
Clock-Gating Naming Conventions.	7-51
Example Script for Naming Style	7-53

Example Script of Output Netlist	7-53
Keeping Clock-Gating Information in a Structural Netlist	7-54
Identifying and Preserving Clock-Gating Cells	7-55
Identification of Clock-Gating Cells.	7-55
Explicit Identification of Clock-Gating Cells	7-55
Preserving the Identified Clock-Gating Cells	7-56
Identified Clock-Gating Cells and dont_touch.	7-58
Handling Clock-Gating Edge Conflicts	7-59
Comparison of Clock-Gate Identification Methods	7-60
Usage Flow With the write_script Command	7-60
Usage Flow With the identify_clock_gating Command.	7-61
Replacing Clock-Gating Cells	7-62
Clock-Gate Optimization Performed During Compilation	7-66
Hierarchical Clock Gating	7-66
Enhanced Register-Based Clock Gating	7-68
Multistage Clock Gating.	7-70
Multistage Clock-Gating Flow.	7-72
Clock Gate Merging	7-72
Placement-Aware Clock Gating in Design Compiler Graphical.	7-75
Clock Gating Multibit Registers	7-76
Performing Clock-Gating on DesignWare Components	7-77
Reporting Command for Clock Gates.	7-78
The report_clock_gating Command.	7-78
8. XOR Self-Gating	
XOR Self-Gating Concepts.	8-2
Using XOR Self-Gating in Power Compiler	8-3
XOR Self-Gating Flows	8-3
Library Requirements for XOR Self-Gating	8-5
Unsupported Registers for XOR Self-Gating.	8-6
Sharing XOR Self-Gates.	8-6
Inserting XOR Self-Gates	8-6
Specifying Objects for XOR Self-Gating	8-7
XOR Self-Gating the Clock-Gated Registers	8-7
Specifying Options for XOR Self-Gating	8-7

Querying the XOR Self-Gates	8-8
Reporting the XOR Self-Gates	8-8
9. Power Optimization	
Overview	9-2
Input and Output of Power Optimization	9-2
Power Optimization in Synthesis Flow	9-4
General Gate-Level Power Optimization	9-5
Leakage Power Optimization	9-5
Dynamic Power Optimization	9-5
Enabling Power Optimization	9-6
Leakage Power Optimization Based on Threshold Voltage	9-7
Multiple Threshold Voltage Library Attributes	9-7
The set_multi_vth_constraint Command	9-8
Analyzing the Multiple Threshold Voltage Library Cells	9-8
Leakage Optimization for Multicorner-Multimode Designs	9-10
Performing Power Optimization	9-11
Settings for Power Optimization	9-11
Power Optimization in the Synopsys Physical Guidance Flow	9-11
Settings for Low-Power Placement	9-12
10. Multivoltage Design Concepts	
Multivoltage and Multisupply Designs	10-2
Library Requirements for Multivoltage Designs	10-2
Liberty PG Pin Syntax	10-3
Level-Shifter Cells	10-3
PG Pin Configuration Support	10-3
Isolation Cells	10-4
Using Standard Cells as Isolation Cells	10-5
Single-Rail and Dual-Rail Isolation Cells	10-5
NOR-Style Isolation Cells	10-6
Requirements of Level-Shifter and Isolation Cells	10-6
Retention Register Cells	10-7
Multithreshold-CMOS Retention Registers	10-7
Power-Switch Cells	10-9

Always-On Logic Cells	10-9
Power Domains	10-10
Shut-Down Blocks	10-11
Marking Pass-Gate Library Pins	10-11
Voltage Areas	10-12
11. UPF Multivoltage Design Implementation	
Multivoltage Design Flow Using UPF	11-3
Power Intent Concepts	11-5
UPF Script Example	11-9
Defining Power Intent in UPF	11-11
Name Spacing Rules for UPF Objects and Attributes	11-12
Defining the Power Intent in the GUI	11-12
UPF Diagram View	11-14
Creating Power Domains	11-14
Representation of Power Domain in the UPF Diagram View	11-16
Scope	11-16
Expanding and Collapsing Power Domains in the GUI	11-17
Viewing Hierarchical Cell and Power Domain Boundaries	11-18
Creating Supply Ports	11-20
Adding Port State Information to Supply Ports	11-21
Representation of Supply Ports in the UPF Diagram View	11-22
Creating Supply Nets	11-22
Specifying Primary Supply Nets for a Power Domain	11-23
Representing Supply Nets in the UPF Diagram View	11-24
Connecting Supply Nets	11-25
Converting the PG Information in RTL to UPF	11-26
Specifying Supply Sets	11-26
Creating Supply Sets	11-27
Creating Supply Set Handles	11-28
Restricting Supply Sets Available to a Power Domain	11-29
Refining Supply Sets	11-29
Associating Supply Sets With Supply Set Handles	11-31

Rules for Associating Supply Sets	11-31
Defining Power States for the Components of a Supply Set	11-32
Correlated Grouping of Supply Voltage Triplets	11-33
Always-On Logic	11-34
Marking Library Cells as Always-On	11-34
Marking Pass-Gate Library Pins	11-34
Always-On Optimization	11-35
Voltage-Aware Always-On Synthesis	11-36
Always-On Optimization on Top-Level Feedthrough Nets	11-36
Always-On Optimization on Disjoint Voltage Area	11-36
Always-On Tie Cells	11-38
Basic Always-On Tie Cell Mapping	11-38
Enhanced Constant Propagation	11-39
Enhanced Always-On Tie Cell Mapping	11-40
Specifying Level-Shifter Strategies	11-40
Using Specific Library Cells With the Level-Shifter Strategy	11-42
Allowing Insertion of Level-Shifters on Clock Nets and Ideal Nets	11-42
Representing Level-Shifter Strategies in the UPF Diagram View	11-42
Specifying Isolation Strategies	11-44
Using the set_isolation_control Command	11-46
Rules Applicable for Location Fanout	11-47
Order of Precedence of Isolation Strategies	11-48
Using Specific Library Cells With Isolation Strategies	11-49
Aligning Isolation Strategies to Constant Drivers	11-49
Isolation and Level-Shifter Cells Connected Back-to-Back	11-52
Representing Isolation Strategies in the UPF Diagram View	11-53
Setting UPF Attributes on Ports and Hierarchical Cells	11-55
Setting Attributes on Ports	11-55
Specifying Supplies for Repeaters	11-57
Setting Attributes on Hierarchical Cells	11-59
Extending the Power Domain Boundary	11-60
Setting Terminal Boundaries	11-60
Specifying Retention Strategies	11-62
Choosing Specific Library Cells With Retention Strategies	11-63
Retention Strategy and Clock-Gating Cells	11-64
Representing Retention Strategies in the UPF Diagram View	11-64

Creating Power Switches	11-65
Representation of Power Switches in the UPF Diagram View.	11-65
Power State Tables	11-66
Creating Power State Tables	11-66
Defining the States of Supply Nets	11-67
Visually Analyzing Power State Tables in the UPF Diagram View.	11-68
Support for Well Bias	11-70
Inserting Power Management Cells	11-70
Reviewing the UPF Specifications	11-72
Commands to Query and Edit Design Objects	11-72
Reviewing the Power Intent Using the Design Vision GUI	11-73
Applying the Power Intent Changes	11-77
Examining and Debugging UPF Specifications	11-78
The check_mv_design Command	11-78
MV Advisor GUI	11-78
Checking for Design Violations.	11-80
Examining Design Violations in the MV Advisor Violation Browser	11-83
Exploring the Violations	11-85
The analyze_mv_design Command	11-89
Analyzing Multivoltage Design Connections in the GUI	11-89
Writing the Power Information.	11-92
Preserving the Command Order in the UPF' File	11-93
Controlling the Line Width in the UPF' File	11-94
Writing and Reading Verilog Netlists With Power and Ground Information	11-95
Power and Ground Supply Connection Syntax	11-95
Supply Sets	11-97
Power Switches	11-97
Reading Verilog Netlists With Power and Ground Supply Connections	11-98
Golden UPF Flow	11-98
Reporting Commands for the UPF Flow.	11-100
UPF-Based Hierarchical Multivoltage Flow Methodology.	11-101
Steps in the Hierarchical UPF Design Methodology	11-101
Block-Level Implementation	11-101
Top-Level Implementation.	11-105

Assembling Your Design	11-106
Characterization of Supply Sets and Supply Nets	11-106
Criteria for Characterization	11-106
Characterization of Supply Sets	11-107
Automatic Inference of Related Supply Net	11-108
Top-Level Design Integration	11-110
Power Domain Merging	11-110
Legacy Blocks	11-111

12. Library Setup for Power Optimization

Basic Library Requirements for Multivoltage Designs	12-2
Power and Ground Pin Syntax	12-2
Converting Libraries to PG Pin Library Format	12-2
Using FRAM View	12-2
Using Tcl Commands	12-3
Tcl Commands for Low-Power Library Specification	12-5
Macro Cells with Fine-Grained Switches	12-5
Library Usage in Multicorner-Multimode Designs	12-6
Link Libraries With Equal Nominal PVT Values	12-6
Setting the dont_use Attribute on Library Cells	12-8
Distinct PVT Requirements	12-8
Automatic Detection of Driving Cell Library	12-10
Relating the Minimum Library to the Maximum Library	12-11
Unique Identification of Libraries Based on File Names	12-11
Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells	12-12
Using the set_opcond_inference Command	12-12
Deviating from the Inferred Operating Condition and Its Impact	12-13

13. Power Optimization in Multicorner-Multimode Designs

Optimizing Multicorner-Multimode Designs	13-2
Optimizing for Leakage Power	13-2
Optimizing for Dynamic Power Using Low-Power Placement	13-4
Reporting Commands	13-5
report_scenarios Command	13-5
Reporting Examples for Multicorner-Multimode Designs	13-6
Script Example for Multicorner-Multimode Flow	13-8

Appendix A. Lower Domain Boundary Support

Introduction	A-2
Enabling the Lower Domain Boundary Feature	A-2
Changes to the Application of the Isolation and Level-Shifter Strategies	A-3
Specifying Design Instances Using Wildcard Characters	A-4
Specifying Design Instances Using SystemVerilog Elements.	A-5
Filtering the Design Elements Using the -applies_to Option	A-6
Insertion of Back-to-Back Isolation and Level-Shifter Cells	A-6
Impact on Hierarchical Flow	A-7
Bottom-Up Flow	A-8
Top-Down Flow	A-9
Characterization of the Related Supply	A-10

Appendix B. Integrated Clock-Gating Cell Example

Library Description	B-2
Example Schematics	B-4
Rising-Edge Latch-Based Integrated Cells	B-5
Rising-Edge Latch-Free Integrated Cells.	B-7
Falling Edge Latch-Based Integrated Cells	B-8
Falling-Edge Latch-Free Integrated Cells	B-10

Appendix C. Attributes for Querying and Filtering

Derived Attribute Lists.	C-2
Usage Examples.	C-4

Appendix D. Power Compiler Command and Variable Reference

Getting Help	D-2
Accessing Help	D-2
Man Page Viewing Instructions.	D-2
Viewing Man Pages in SolvNet	D-2
Setting Up the UNIX Environment	D-3
Viewing Man Pages From UNIX	D-3
Viewing Man Pages From dc_shell	D-3
Power Compiler Commands.	D-3

add_power_state	D-3
add_pst_state	D-4
all_clock_gates	D-4
all_isolation_cells	D-4
all_level_shifters	D-4
all_self_gates	D-5
all_upf_repeater_cells	D-5
analyze_library	D-5
analyze_dw_power	D-5
analyze_mv_design	D-5
apply_clock_gate_latency	D-6
associate_supply_set	D-6
characterize	D-6
check_level_shifters	D-6
check_mv_design	D-6
compile	D-7
compile_ultra	D-8
connect_logic_net	D-8
connect_supply_net	D-8
convert_pg	D-8
create_logic_net	D-9
create_logic_port	D-9
create_power_domain	D-9
create_power_state_group	D-9
create_power_switch	D-9
create_pst	D-10
create_supply_net	D-10
create_supply_port	D-10
create_supply_set	D-10
find_objects	D-10
get_power_domains	D-11
get_power_switches	D-11
get_related_supply_net	D-11
get_supply_nets	D-11
get_supply_ports	D-12
generate_mv_constraints	D-12

identify_clock_gating	D-12
infer_switching_activity	D-13
insert_clock_gating	D-13
insert_isolation_cell	D-13
insert_mv_cells	D-13
lib2saif	D-14
load_upf	D-14
map_isolation_cell	D-14
map_level_shifter_cell	D-14
map_power_switch	D-14
map_retention_cell	D-15
merge_saif	D-15
propagate_constraints	D-15
propagate_switching_activity	D-16
query_cell_instances	D-16
query_cell_mapped	D-16
query_map_power_switch	D-17
query_net_ports	D-17
query_port_net	D-17
query_port_state	D-17
query_power_switch	D-17
query_pst	D-17
query_pst_state	D-18
read_saif	D-18
remove_clock_gating	D-18
remove_clock_gating_style	D-19
remove_dft_clock_gating_pin	D-19
remove_isolation_cell	D-19
remove_level_shifters	D-19
remove_power_domain	D-19
remove_upf	D-19
replace_clock_gates	D-20
report_clock_gating	D-20
report_dft_clock_gating_configuration	D-20
report_dft_clock_gating_pin	D-20
report_isolation_cell	D-21

report_level_shifter	D-21
report_lib	D-21
report_mv_library_cells	D-22
report_power	D-22
report_power_calculation	D-23
report_power_domain	D-23
report_power_gating	D-23
report_power_pin_info	D-23
report_power_switch	D-23
report_pst	D-24
report_retention_cell	D-24
report_saif	D-24
report_self_gating	D-24
report_supply_net	D-25
report_supply_port	D-25
reset_clock_gate_latency	D-25
reset_dft_clock_gating_configuration	D-25
reset_switching_activity	D-25
rewire_clock_gating	D-25
saif_map	D-26
save_upf	D-26
set_cell_internal_power	D-27
set_clock_gate_latency	D-27
set_clock_gating_objects	D-27
set_clock_gating_registers	D-27
set_clock_gating_enable	D-28
set_clock_gating_style	D-28
set_cost_priority	D-28
set_design_attributes	D-29
set_dft_clock_gating_configuration	D-29
set_dft_clock_gating_pin	D-29
set_dft_power_control	D-29
set_domain_supply_net	D-29
set_dont_use	D-30
set_equivalent	D-30
set_isolation	D-30

set_isolation_cell	D-30
set_isolation_control	D-31
set_leakage_power_model	D-31
set_level_shifter	D-31
set_level_shifter_cell	D-31
set_multi_vth_constraint	D-32
set_port_attributes	D-32
set_power_clock_scaling	D-32
set_power_guide	D-33
set_power_prediction	D-33
set_power_switch_cell	D-33
set_query_rules	D-33
set_related_supply_net	D-34
set_replace_clock_gates	D-34
set_retention	D-34
set_retention_cell	D-35
set_retention_control	D-35
set_retention_control_pins	D-35
set_retention_elements	D-36
set_scenario_options	D-36
set_scope	D-36
set_self_gating_objects	D-36
set_self_gating_options	D-36
set_switching_activity	D-37
set_switching_activity_profile	D-37
set_upf_query_options	D-37
unset_power_guide	D-37
upf_version	D-38
write_saif	D-38
write_script	D-38
Power Compiler Variables	D-38
abstraction_enable_power_calculation	D-38
compile_power_domain_boundary_optimization	D-39
enable_golden_upf	D-39
enable_rule_based_query	D-39
golden_upf_report_missing_objects	D-39

hdlin_enable_upf_compatible_naming	D-39
link_allow_upf_design_mismatch	D-39
mv_allow_ls_on_leaf_pin_boundary	D-40
mv_allow_va_beyond_core_area	D-40
mv_input_enforce_simple_names	D-40
mv_insert_level_shifters_on_ideal_nets	D-40
mv_make_primary_supply_available_for_always_on	D-40
mv_no_always_on_buffer_for_redundant_isolation	D-40
mv_no_cells_at_default_va	D-41
mv_no_main_power_violations	D-41
mv_output_enforce_simple_names	D-41
mv_output_upf_line_indent	D-41
mv_output_upf_line_width	D-41
mv_skip_opcond_checking_for_unloaded_level_shifter	D-41
mv_upf_tracking	D-42
mv_use_std_cell_for_isolation	D-42
physopt_power_critical_range	D-42
power_cg_all_registers	D-42
power_cg_auto_identify	D-42
power_cg_balance_stages	D-42
power_cg_cell_naming_style	D-43
power_cg_derive_related_clock	D-43
power_cg_designware	D-43
power_cg_enable_alternative_algorithm	D-43
power_cg_ext_feedback_loop	D-43
power_cg_flatten	D-43
power_cg_gated_clock_net_naming_style	D-44
power_cg_ignore_setup_condition	D-44
power_cg_inherit_timing_exceptions	D-44
power_cg_iscgs_enable	D-44
power_cg_module_naming_style	D-44
power_cg_physically_aware_cg	D-44
power_cg_print_enable_conditions	D-45
power_cg_print_enable_conditions_max_terms	D-45
power_cg_reconfig_stages	D-45
power_cg_sequential_clock_gating	D-45

power_default_static_probability	D-45
power_default_toggle_rate	D-45
power_default_toggle_rate_type	D-46
power_do_not_size_icg_cells	D-46
power_enable_clock_scaling	D-46
power_enable_datapath_gating	D-46
power_enable_one_pass_power_gating	D-46
power_enable_power_gating	D-46
power_fix_sdpd_annotation	D-47
power_fix_sdpd_annotation_verbose	D-47
power_hdlc_do_not_split_cg_cells	D-47
power_keep_license_after_power_commands	D-47
power_lib2saif_rise_fall_pd	D-47
power_low_power_placement	D-47
power_min_internal_power_threshold	D-48
power_model_preference	D-48
power_opto_extra_high_dynamic_power_effort	D-48
power_preserve_rtl_hier_names	D-48
power_rclock_inputs_use_clocks_fanout	D-48
power_rclock_unrelated_use_fastest	D-48
power_rclock_use_asynch_inputs	D-49
power_remove_redundant_clock_gates	D-49
power_rtl_saif_file	D-49
power_sa_propagation_verbose	D-49
power_same_switching_activity_on_connected_objects	D-49
power_sdpd_message_tolerance	D-49
synlib_enable_analyze_dw_power	D-50
upf_allow_DD_primary_with_supply_sets	D-50
upf_allow_refer_before_define	D-50
upf_auto_iso_clamp_value	D-50
upf_auto_iso_enable_source	D-50
upf_auto_iso_isolation_sense	D-50
upf_block_partition	D-51
upf_charz_allow_port_punch	D-51
upf_charz_enable_supply_port_punching	D-51
upf_charz_max_srsn_messages	D-51

upf_create_implicit_supply_sets	D-51
upf_enable_legacy_block	D-51
upf_enable_relaxed_charz	D-52
upf_extension	D-52
upf_isols_allow_instances_in_elements	D-52
upf_iso_filter_elements_with_applies_to	D-52
upf_levshi_on_constraint_only	D-52
upf_name_map	D-52
upf_report_isolation_matching	D-53
upf_skip_ao_check_for_els_input	D-53
upf_suppress_etm_model_checking	D-53
upf_suppress_message_in_black_box	D-53
upf_suppress_message_in_etm	D-53

Preface

This preface includes the following sections:

- [About This User Guide](#)
- [Customer Support](#)

About This User Guide

This user guide describes the Power Compiler tool, its methodology, and its use. Power Compiler is a comprehensive tool that assists you in analysis and optimization of your design for power.

Audience

The *Power Compiler User Guide* builds on concepts introduced in Design Compiler publications. It is assumed in this user guide that the user has some familiarity with Design Compiler products.

Related Publications

For additional information about Power Compiler, see the documentation on the Synopsys SolvNet® online support site at the following address:

<https://solvnet.synopsys.com/DocsOnWeb>

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- DFT Compiler
- Formality®
- PrimeTime® PX

Release Notes

Information about new features, changes, enhancements, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *Power Compiler Release Notes* in SolvNet.

To see the *Power Compiler Release Notes*,

1. Go to the Download Center on SolvNet located at the following address:
<https://solvnet.synopsys.com/DownloadCenter>
2. Select Power Compiler, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as prompt> write_file top
[]	Denotes optional arguments in syntax, such as <code>write_file [-format <i>fmt</i>]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</code>
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

<https://solvnet.synopsys.com>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at <https://solvnet.synopsys.com>, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>
- Telephone your local support center.
 - Call (800) 245-8005 from within North America.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>

Part 1: Power Compiler Concepts

1

Introduction to Power Compiler

This chapter describes the Power Compiler methodology and describes power library models and power analysis technology. In addition, it provides library and license requirements.

Power Compiler is part of Synopsys Design Compiler synthesis family. It performs both RTL and gate-level power optimization and gate-level power analysis. By applying Power Compiler's various power reduction techniques, including clock-gating, multivoltage leakage power optimization, and gate-level power optimization, you can achieve power savings, and area and timing optimization in the front-end synthesis domain.

This chapter contains the following sections:

- [Power Compiler Methodology](#)
- [Power Library Models](#)
- [Power Analysis Technology](#)
- [Power Optimization Technology](#)
- [Getting Started With Power Compiler](#)

Power Compiler Methodology

With the increasing popularity of portable-oriented applications, low-power designs have become crucial elements for product success. Most especially, the static power (leakage power) consumption concern is more pronounced as the technology becomes more smaller and faster.

Power Compiler tool provides a complete methodology for low-power designs, which consist of the following:

- Power analysis

The tool analyzes the design and computes average power consumption based on the switching activity of the nets.

You can perform power analysis at the following levels:

- Register transfer level using RTL simulation
- Gate level using RTL or gate-level simulation

- Power optimization

The power optimization technology optimizes your design for power consumption. It computes average power consumption based on the activity of the nets in your design.

The tool performs the following types of power optimization:

- Leakage power or static power optimization
 - Multivoltage threshold power optimization
 - Power switching
- Dynamic power optimization
 - Clock-gating
 - XOR self-gating
 - Low power placement
- Multivoltage and multicorner-multimode support

Power Library Models

The power library model analyzes leakage, switching, and internal power.

For more information about library modeling and characterization for power, see the Library Compiler documentation.

The Power Compiler gate-level power model supports the following features:

- Composite Current Source (CCS) library support
- Lookup tables based on output pin capacitance and input transition time
- Cells with multiple output pins
- State-dependent and path-dependent internal power
- Leakage power, including state-dependent and path-dependent internal power
- Separate specification of rise and fall power in the internal power group

In addition, you can use the CCS power model. CCS models represent the physical circuit properties more closely than other models to the simulated data obtained during characterization with SPICE. It is a current-based power model that contains the following features:

- One library format suitable for a wide range of applications, including power analysis and optimization.
- Power analysis with much higher time resolution compared to NLPM.
- Dynamic power characterized by current waveforms stored in the library. The charge can be derived from the current waveform.
- Leakage power modeled as the actual leakage current. The leakage current does not artificially depend on the reference voltage, as is the case with leakage power. This facilitates voltage scaling.
- Standard-cell and macro modeling.

Power Analysis Technology

Power Compiler analyzes your design for net switching power, cell internal power, and leakage power. Power Compiler also enables you to perform power analysis of your gate-level design using switching activity from RTL or gate-level simulation or user-annotation.

When analyzing a gate-level design, Power Compiler requires a gate-level netlist and switching activity for the netlist. Using steps described in this book, Power Compiler enables you to capture the switching activity of primary inputs, primary outputs, and outputs of sequential elements during RTL simulation. After you annotate the captured activity on design elements, Power Compiler propagates switching activity through the nonannotated portions of your design.

Using power analysis by way of switching activity from RTL simulation provides a much faster turnaround than analysis using switching activity by way of gate-level simulation.

If you require more accuracy during the later stages of design development, you can annotate some or all of the nets of your design with switching activity from full-timing gate-level simulation.

Power Compiler supports the following power analysis features:

- Performs gate-level power analysis.
- Analyzes net switching power, cell internal power, and leakage power.
- Accepts input as either user-defined switching activity, switching activity from RTL or gate-level simulation, or a combination of both. The default is vector-free.
- Propagates switching activity during power analysis to nonannotated nets.
- Supports sequential, hierarchical, gated clock, and multiple-clock designs.
- Supports RAM and I/O modeling using a detailed state-dependent and path-dependent power model.
- Performs power analysis in a single, integrated environment at multiple phases of the design process.
- Reports power at any level of hierarchy to enable quick debugging.
- Reports capability to validate your testbench.
- Supports interfaces to NC-Sim, MTI, VCS-MX, and Verilog-XL simulators for toggle data.

Synopsys also provides another gate level detail power analysis tool called PrimeTime PX. PrimeTime PX can analyze peak power, glitch power and X-state power. It also has time based power waveform and supports special modes of operation.

For more information, see the *PrimeTime PX User Guide*.

Power Optimization Technology

You can optimize your design for power using the following capabilities:

- RTL clock gating
- Gate-level multivoltage and dynamic power optimization

RTL clock gating is the most effective power optimization feature provided by Power Compiler. This is a high-level optimization technique that can save a significant amount of power by adding clock gates to registers that are not always enabled and with synchronous load-enable or explicit feedback loops. This greatly reduces the power consumption of the design by reducing switching activity on the clock inputs to registers and eliminating the multiplexers. It also results in a lower area consumption for your design.

RTL clock gating optimizes for dynamic power and can be applied simultaneously on a design.

When a gate-level power optimization constraint is set in the design, by default, Power Compiler performs optimization to meet the constraints for design rule checking, timing, power and area in that order of priority.

The Power Compiler gate-level power optimization solution offers the following features:

- Push-button user interface to reduce power consumption
- Multivoltage libraries for leakage optimization with short turnaround time
- Simultaneous optimization for timing, power, and area
- Optimization based on circuit activity, capacitance, and transition times
- Power analysis capability; optimizes with the same detailed power library model used in analysis
- Operates within Galaxy platform and is compatible with other Synopsys tools (Design Compiler, Floorplan Manager, Module Compiler, DFT Compiler, and Formality)

Getting Started With Power Compiler

This section provides information about the basic requirements to analyze and optimize for power.

Library Requirements

Power Compiler uses technology libraries characterized for power. You can characterize your library with the following power features:

Internal Power

To optimize for dynamic power, Power Compiler requires libraries characterized for internal power. This is the minimum library requirement to characterize for power. This characteristic accounts for short-circuit power consumed internal to gates.

Leakage Power

To optimize for static power, Power Compiler requires libraries characterized for leakage power. This characteristic accounts for the power dissipated while the device is not in use. Power Compiler also supports multivoltage libraries.

State and Path Dependency

To optimize for varying modes of operation, Power Compiler requires libraries characterized for state-dependency. To optimize for varying power consumption based on various input to output paths, Power Compiler requires libraries characterized for path-dependency.

To capture state-dependent and path-dependent switching activity from simulation, library cells must have state- and path- dependent information in the lookup tables for internal power and pin capacitance. Synopsys Power Compiler uses state-dependent and path-dependent switching activity to compute state-dependent and path-dependent switching power.

If you are developing libraries to use with Synopsys power products, see the Library Compiler documentation. Power Compiler supports non-linear power models, scalable polynomial equation power models, and composite current source libraries.

Command-Line Interface

Power Compiler is accessible from the Design Compiler command-line interface if you have an appropriate license. See [“License Requirements”](#).

Using the Design Compiler command-line interface, power optimization takes place during your `dc_shell` optimization session. For more information about its command-line interface, see the Design Compiler documentation.

Power Compiler also works within the Design Compiler topographical domain shell (`dc_shell-topo`). Whereas `dc_shell` uses wide-load models for timing and area power optimizations, `dc_shell-topo` uses placement timing values instead. For more information, see the Design Compiler documentation.

Note:

Unless otherwise noted, all functionality described in this manual pertains to both `dc_shell` and `dc_shell-topo`. Also unless otherwise noted, this manual uses "`dc_shell`" as a generic term that applies to the Design Compiler topographical domain also.

Graphical User Interface

Power Compiler is accessible from Design Vision, the graphical user interface (GUI) for the Synopsys logic synthesis environment. You must have the Design Vision license and other appropriate licenses to perform power analysis and optimizations. For more details, see [“License Requirements”](#).

Design Vision supports menus and dialog boxes for the frequently used synthesis features. The Power menu in the GUI allows you to specify, modify, and review your power architecture.

For more details on specifying power intent using the GUI, see [Chapter 11, “UPF Multivoltage Design Implementation](#). For details about general usage of Design Vision, see the *Design Vision User Guide*.

License Requirements

Power analysis and optimization using Power Compiler require either one of the following two combination of licenses:

- Power-Optimization
- Power-Analysis + Power-Optimization-Upgrade

These licenses also allow you to perform multivoltage power optimization and analysis.

Power Compiler is incorporated within Design Compiler. You need the license for Design Compiler in addition to the power licenses.

Design Vision License

You can also perform power analysis and power optimizations using the Design Vision GUI. To use Design Vision, you need the Design-Vision license. To use Design Vision in topographical mode, you need a Design-Vision license, a DesignWare license and the DC Ultra package.

How the Licenses Work

When you invoke `dc_shell`, no power license is checked out until you use a Power Compiler feature. When the Power Compiler feature is completed, your power license is released.

Synopsys licensing software and the documentation describing it are separate from the tools that use it. You install, configure, and use a single copy of Synopsys Common Licensing (SCL) for all Synopsys tools. Because SCL provides a single, common licensing base for all Synopsys tools, it reduces licensing administration complexity, minimizing the effort you expend in installing, maintaining, and managing licensing software for Synopsys tools.

For complete Synopsys licensing information, see the *Synopsys Common Licensing Administration Guide*. This guide provides detailed information about SCL installation and configuration, including examples of license key files and troubleshooting guidelines.

Reading and Writing Designs

When using `dc_shell`, you read designs from disk before working on them, make changes to them, and write them back to the disk.

Power Compiler can read or write a gate-level netlist in any of the formats shown in [Table 1-1](#).

Table 1-1 File Formats and Extensions

Format	Default extension	File type	Special license key required?
db	.db	Synopsys internal database format	No
ddc	.ddc	Synopsys Design Compiler database format (the default)	No
equation	.eqn	Synopsys equation format	No
LSI	.NET	LSI Logic Corporation netlist format	Yes
MENTOR	.neted	Mentor intermediate netlist format (see <i>Synopsys Mentor Interface Application Note</i>)	Yes
PLA	.pla	Berkeley (Espresso) PLA format	No
ST	.st	Synopsys state table format	No
TDL	.tdl	Tegas Design Language (TDL) netlist format	Yes
Verilog	.v	Hardware Description Language	Yes
VHDL	.vhd	VHSIC Hardware Description	Yes

Note:

NLPM and CCS are the supported power models in the library.

2

Power Compiler Design Flow

As you create a design, it moves from a high level of abstraction to its final implementation at the gate level. Power Compiler offers analysis and optimization throughout the design cycle, from RTL to the gate level.

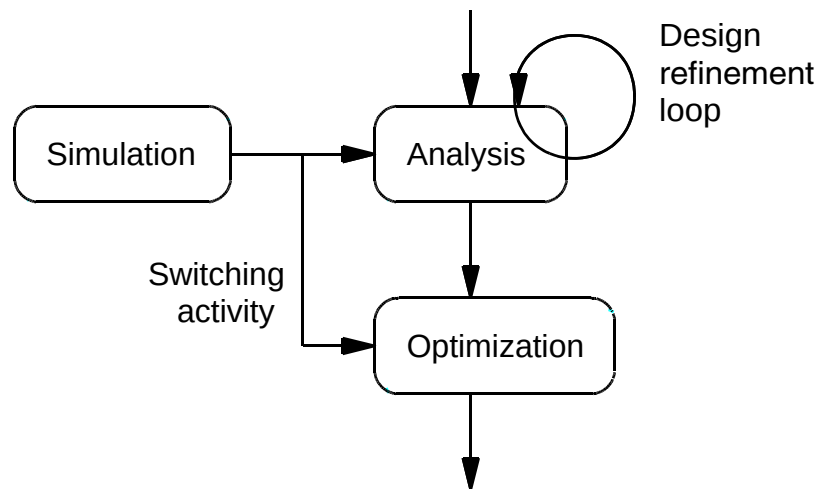
This chapter contains the following sections:

- [Power in the Design Cycle](#)
- [Power Optimization and Analysis Flow](#)

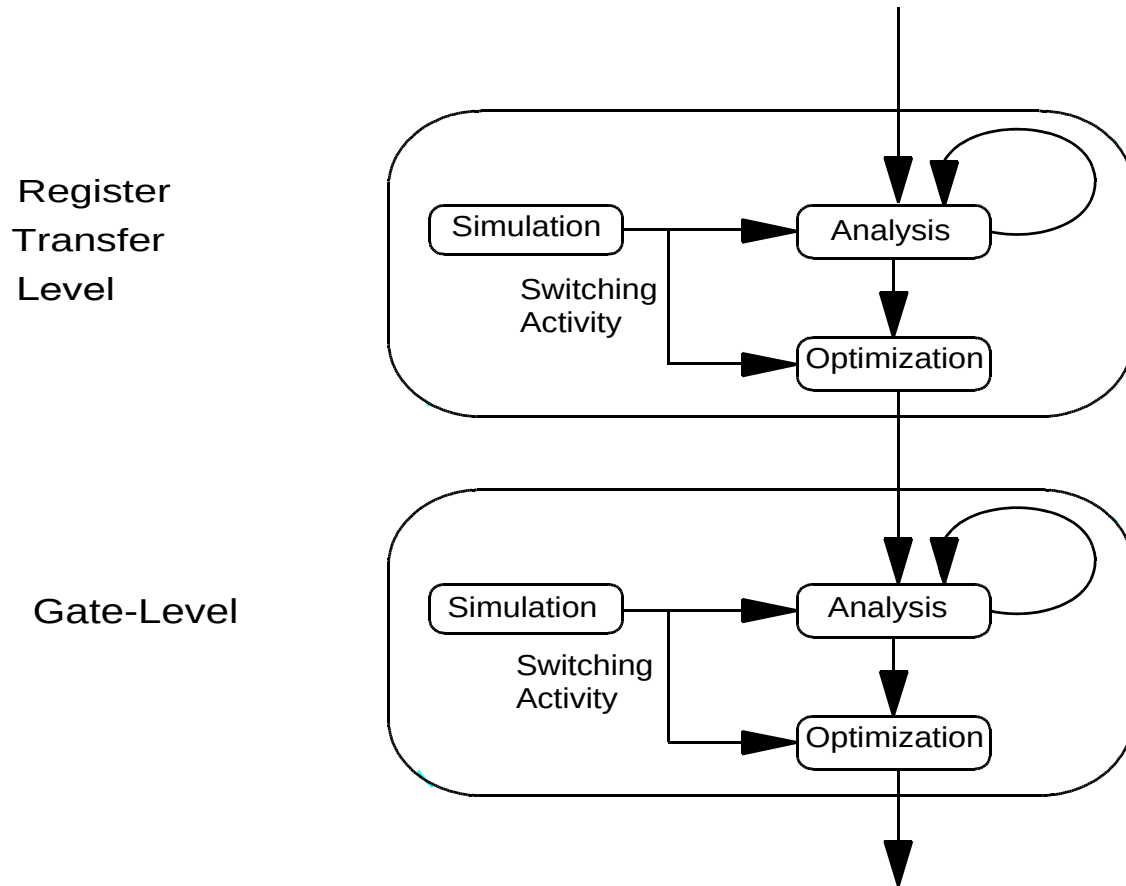
Power in the Design Cycle

At each level of abstraction, use simulation, analysis, and optimization to refine your design before moving to the next lower level of design abstraction. The relationship of these three processes is shown in [Figure 2-1](#).

Figure 2-1 Power Flow at Each Abstraction Level



Simulation, analysis, and optimization occur at each level of abstraction. Design refinement loops occur within each level. Simulation and the resultant switching activity give analysis and optimization the necessary information to refine the design before going to the next lower level of abstraction. The entire flow is shown in [Figure 2-2](#).

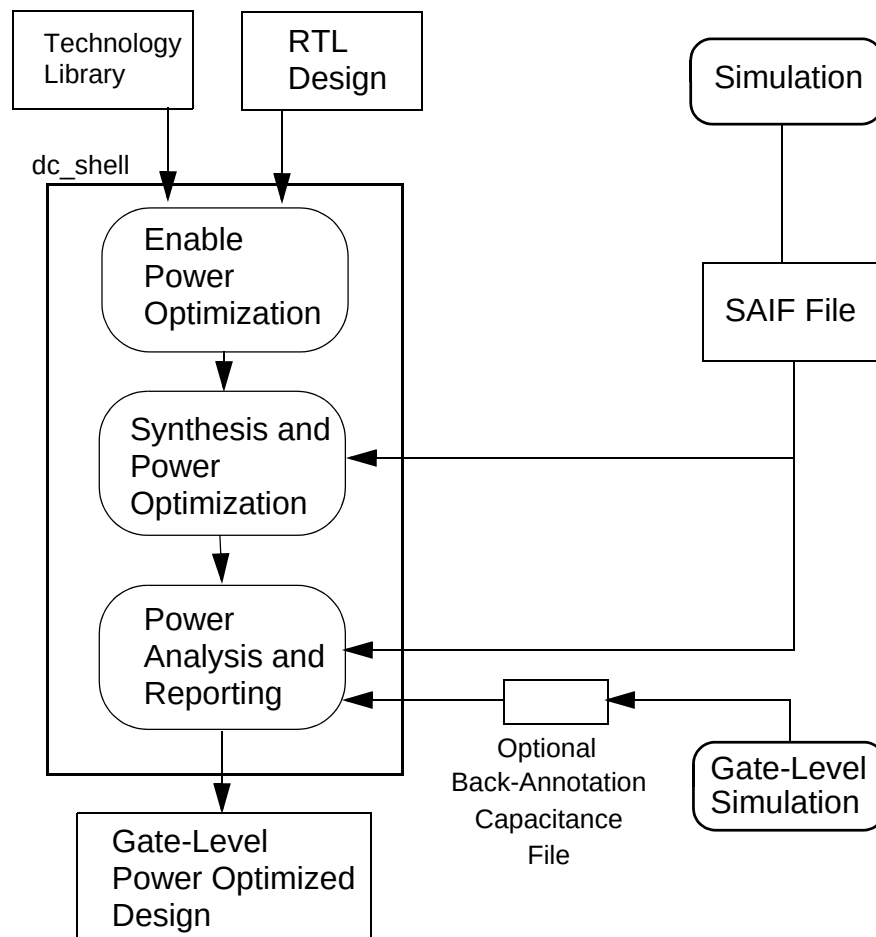
Figure 2-2 Power Flow From RTL to Gate-Level

Using Power Compiler, you can analyze and optimize at the RTL and gate levels. The higher the level of design abstraction, the greater the power savings you can achieve.

Power Optimization and Analysis Flow

Figure 2-3 shows a high-level power optimization and analysis flow.

Figure 2-3 Power Optimization and Analysis Flow



The power optimization starts with the specified RTL design and logic library and results in a power-optimized gate-level netlist.

During analysis and optimization, Power Compiler uses information in the logic library. To optimize or analyze dynamic power and leakage power, the logic library must be characterized for internal power. To optimize or analyze static power, the logic library must be characterized for leakage power.

You can use Power Compiler to analyze the gate-level netlist produced by Design Compiler or the power-optimized netlist produced by Power Compiler.

Simulation

Most of the steps in the flow occur within the Design Compiler environment, `dc_shell`. However, [Figure 2-3](#) shows that the power flow requires a SAIF file, which is generated by simulation.

Simulation generates information about the design's switching activity and creates a Switching Activity Information Format (SAIF) file, which is used for annotation purposes. For information, see [Chapter 4, “Generating SAIF Files.”](#)

During power analysis, Power Compiler uses annotated switching activity to evaluate the power consumption of your design. During power optimization, Power Compiler uses annotated switching activity to make optimization decisions about your design. For information, see [Chapter 5, “Annotating Switching Activity.”](#)

Enable Power Optimization

Power Compiler provides several techniques for optimizing power, such as clock gating and operand isolation. Power optimization achieved at higher levels of abstraction has an increasingly important impact on reduction of power in the final gate-level implementation. You enable power optimizations with Power Compiler commands described in this manual.

Synthesis and Power Optimization

Design Compiler and Power Compiler work together within the `dc_shell` environment to synthesize your design to a gate-level netlist optimized for power. Synthesis with power optimization occurs during Design Compiler's compile processing.

In the Synopsys physical guidance flow, the tool can perform low-power placement to reduce the dynamic power consumption of the design. For more details, see [“Power Optimization in the Synopsys Physical Guidance Flow” in Chapter 9.](#)

Power Analysis and Reporting

You can use Power Compiler for analysis of your gate-level design at several points in your methodology flow. [Figure 2-3](#) shows power analysis after power optimization, which results in a detailed report of your power-optimized netlist.

You can also analyze power before synthesis and power optimization. For example, after annotating the switching activity from your SAIF file to verify that the annotation is correct. Analysis before power optimization provides an optional reference point for comparison with the power-optimized netlist.

3

Power Modeling and Calculation

As you create a design, it moves from a high level of abstraction to its final implementation at the gate level. Power Compiler supports analysis and optimization throughout the design cycle, from RTL to the gate level.

This chapter contains the following sections:

- [Power Types](#)
- [Calculating Power](#)
- [Using CCS Power Libraries](#)
- [Voltage Scaling](#)

Power Types

The power dissipated in a circuit falls into two broad categories:

- Static power
- Dynamic power

Static Power

Static power is the power dissipated by a gate when it is not switching, that is, when it is inactive or static.

Static power is dissipated in several ways. The largest percentage of static power results from source-to-drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called leakage power.

Dynamic Power

Dynamic power is the power dissipated when the circuit is active. A circuit is active anytime the voltage on a net changes due to some stimulus applied to the circuit. Because voltage on an input net can change without necessarily resulting in a logic transition on the output, dynamic power can be dissipated even when an output net does not change its logic state.

The dynamic power of a circuit is composed of two kinds of power:

- Switching power
- Internal power

Switching Power

The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output.

Because such charging and discharging are the result of the logic transitions at the output of the cell, switching power increases as logic transitions increase. Therefore, the switching power of a cell is a function of both the total load capacitance at the cell output and the rate of logic transitions.

Internal Power

Internal power is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.

To illustrate the cause of short-circuit power, consider the simple gate shown in [Figure 3-1](#). A rising signal is applied at IN. As the signal transitions from low to high, the N type transistor turns on and the P type transistor turns off. However, for a short time during signal transition, both the P and N type transistors can be on simultaneously. During this time, current I_{sc} flows from V_{dd} to GND, causing the dissipation of short-circuit power (P_{sc}).

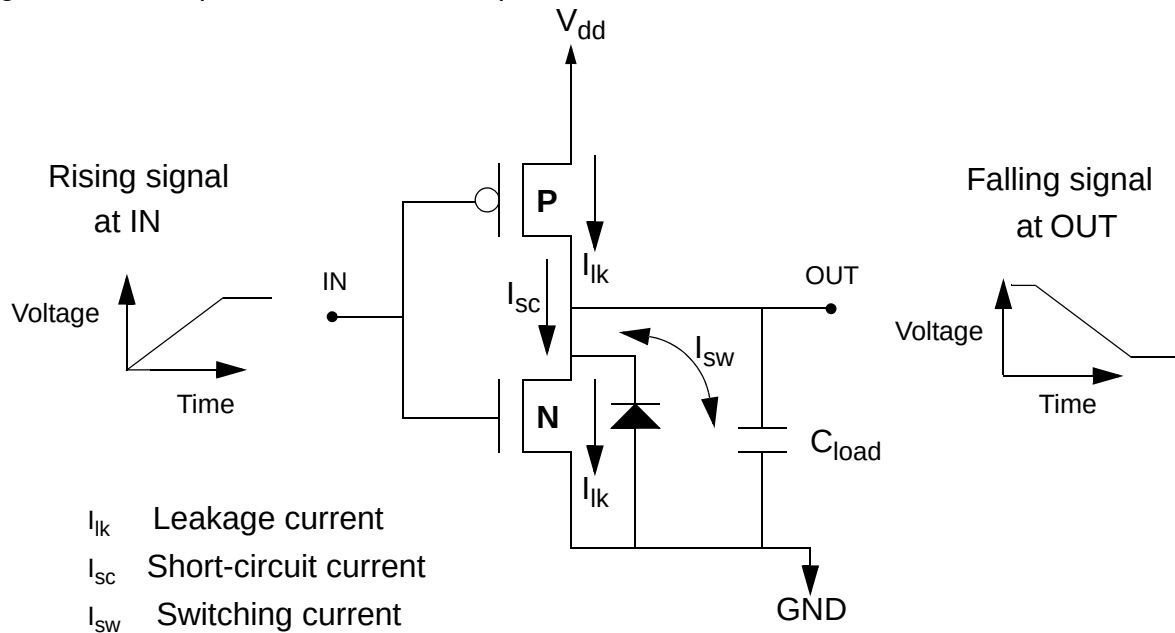
For circuits with fast transition times, short-circuit power can be small. However, for circuits with slow transition times, short-circuit power can account for 30 percent of the total power dissipated by the gate. Short-circuit power is affected by the dimensions of the transistors and the load capacitance at the gate's output.

In most simple library cells, internal power is due mostly to short-circuit power. For more complex cells, the charging and discharging of internal capacitance might be the dominant source of internal power.

Library developers can model internal power by using the internal power library group. For more information about modeling internal power, see the *Library Compiler User Guide*.

[Figure 3-1](#) shows a simple gate and illustrates where static and dynamic power are dissipated.

Figure 3-1 Components of Power Dissipation



Calculating Power

Power analysis calculates and reports power based on the equations that accompany this chapter. Power Compiler uses these equations and the information modeled in the specified logic library to evaluate the power of your design. This chapter includes information about library modeling for power where equations for power types appear.

For more information about modeling power in your library, see the *Library Compiler User Guide*.

Note:

The power calculations described in this section only apply to NLPM power calculations.

Leakage Power Calculation

Power Compiler analysis computes the total leakage power of a design by summing the leakage power of the design's library cells, as shown in the following equation:

$$P_{\text{LeakageTotal}} = \sum_{\forall \text{cells}(i)} P_{\text{CellLeakage}_i}$$

Where:

$P_{\text{LeakageTotal}}$ = Total leakage power dissipation of the design

$P_{\text{CellLeakage}_i}$ = Leakage power dissipation of each cell i

Library developers annotate the library cells with appropriate total leakage power dissipated by each library cell. They can provide a single leakage power for all cells in the library by using the `default_cell_leakage_power` attribute or provide leakage power per cell with the `cell_leakage_power` attribute.

If the `cell_leakage_power` attribute is missing or negative, the tool assigns the value of the `default_cell_leakage_power` attribute. If this is not available, Power Compiler assumes default of 0.

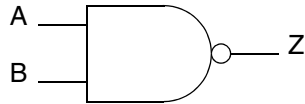
To model state-dependent leakage, use the `leakage_power` attribute. You can also use Boolean expressions to define the conditions for different cell leakage power values.

To calculate cell leakage, Power Compiler determines the units based on the `leakage_power_unit` attribute. It checks for the `leakage_power` attribute first. The leakage value for each state is multiplied by the percentage of the total simulation time at that state and summed to provide the total leakage power per cell.

If the state is not defined in the `leakage_power` attribute, the value of the `cell_leakage_power` attribute is used to obtain the contribution of the leakage power at the undefined state.

[Figure 3-2](#) shows the leakage power calculation performed on a NAND gate with state-dependent values.

Figure 3-2 Leakage Power Calculation for a NAND Gate With State Dependent Values



```

library ....
leakage_power_unit : 1nW ;
cell (NAND) ...
cell_leakage_power : 0.5 ;
leakage_power ( ) {
    when : "A&B"
    value : 0.2
  }
  
```

For a total power consumption time of 600, the cell is at the state defined by the condition A&B for 33% of the time. For the remaining 67% of the simulation time, the default is assumed.

Therefore, the total cell leakage value is:

$$(.33 * .2nW) + (.67 * .5nW) = .4nW$$

Multithreshold Voltage Libraries

Static power dissipation has an exponential dependence on the switching threshold of the transistor's voltage. In order to address low-power designs IC foundries offer technologies that enable multiple threshold voltage libraries.

Each type of logic gate is available in two or more different threshold voltage (vth) groups. The threshold voltage determines the speed and the leakage characteristics of the cell. Cells with low-threshold transistors switch quickly but have higher leakage and consume more power. Cells with high threshold transistors have lower leakage and consume less power but switch more slowly.

For leakage power optimization, Power Compiler supports multiple mechanisms for swapping high and low-threshold voltage cells appropriately, based on power and timing requirements.

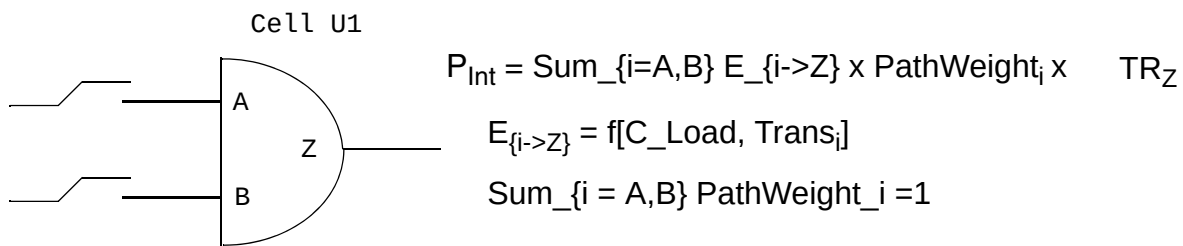
For more details about using the multithreshold voltage libraries, see [Multiple Threshold Voltage Library Attributes](#).

Internal Power Calculation

When computing internal power, power analysis uses information characterized in the logic library. The `internal_power` library group and its associated attributes and groups define scaling factors and a default for internal power. Library developers can use the internal power table to model internal power on any pin of the library cell.

A cell's internal power is the sum of the internal power of all of the cell's inputs and outputs as modeled in the logic library. [Figure 3-3](#) shows how Synopsys power tools calculate the internal power for a simple combinational cell, U1 with path-dependent internal power modeling.

Figure 3-3 Internal Power Model (Combinational)



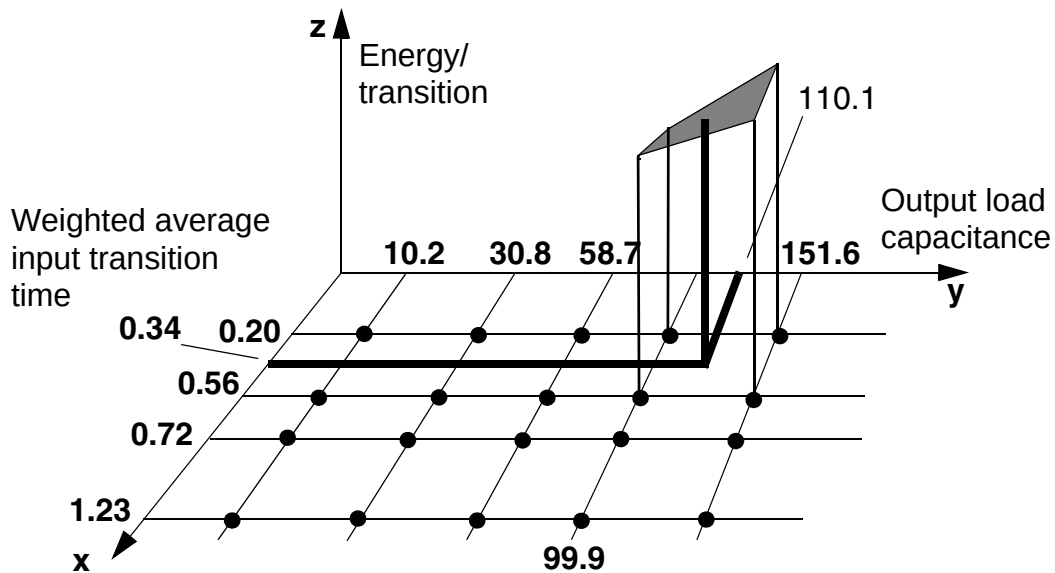
P_{Int}	Total internal power of the cell _E
E_Z	Internal energy for output Z as a function of input transitions, output load, and voltage
TR_Z	Toggle rate of output pin Z, transitions per second
TR_i	Toggle rate of input pin i, transitions per second
$Trans_i$	Transition time of input i
$\text{WeightAvg}_{(Trans)}$	Weighted average transition time for output Z

Power Compiler calculates the input path weights based on the input toggle rates, transition times, and functionality of the cell. Power Compiler supports NLDM (table-based) models.

NLDM Models

To compute the internal power consumption of NLDM models, Power Compiler uses the weighted average transition time as an index to the internal power associated with the output pin. As an additional index to the power table, Power Compiler uses the output load capacitance. The two indexes enable Power Compiler to access the two-dimensional lookup table for the output, as shown in [Figure 3-4](#).

Figure 3-4 Two-Dimensional Lookup Table



For cells in which output pins have equal or opposite logic values, Power Compiler can use a three-dimensional lookup table. Power Compiler indexes the three-dimensional table by using input transition time and both output capacitances of the equal (or opposite) pins. The three-dimensional table is well suited to describing the flip-flop, which has Q and Q-bar outputs of opposite value.

The `internal_power` library group supports a one-, two-, or three-dimensional lookup table. [Table 3-1](#) shows the types of lookup tables, whether they are appropriate to inputs or outputs, and how they are indexed.

Table 3-1 Lookup Tables

Lookup table	Defined on	Indexed by
One-dimensional	Input	Input transition
	Output	Output load capacitance
Two-dimensional	Output	Input transition and output load capacitance
Three-dimensional	Output	Input transition and output load capacitances of two outputs that have equal or opposite logic values

For more information about modeling internal power and library modeling syntax and methodology, see the *Library Compiler User Guide*.

For various operating conditions, the table model supports scaling factors for the internal power calculation. These are,

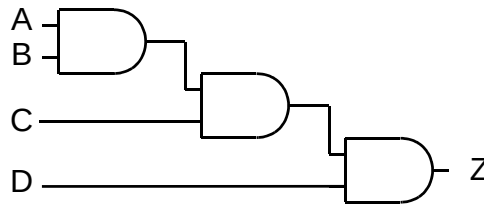
- `k_process_internal_power`
- `k_temp_internal_power`
- `k_volt_internal_power`

These factors however do not accurately model the non-linear effects of the operating conditions, so most vendors generate separate table-based libraries for different operating conditions.

State and Path Dependency

Cells often consume different amounts of internal power, depending on which input pin transitions or depending on the state of the cell. These are state and path dependent.

To demonstrate path-dependent internal power, consider the following simple library cell, which has several levels of logic and a number of input pins:



Input A and input D can each cause an output transition at Z. However, input D affects only one level of logic, whereas input A affects all three. An output transition at Z consumes more internal power when it results from an input transition at A than when it results from an input transition at D. You can specify multiple lookup tables for outputs, depending on the input transitions.

Power Compiler chooses the appropriate path dependent internal power table for an output by checking the `related_pin` attribute in the library. Based on the percentage of toggles on each input pin, the total power due to transitions on the output pin is calculated by accessing the correct table or equation for each related pin and applying the percentage contribution per input pin.

An example of a cell with state-dependent internal power is a RAM cell. It consumes a different amount of internal power depending on whether it is in read or write model. You can specify separate tables or equations depending on the state or mode of the cell.

If the toggle rate information is provided for each state defined in the power model, Power Compiler accesses the appropriate information. If only the input or output toggle information is available, Power Compiler averages the tables for the different states to compute the internal power of the cell.

For more information about how the toggle information affects the internal power analysis, see [Performing Power Analysis](#).

Rise and Fall Power

When a signal transitions, the internal power related to the rising transition is different from the internal power related to the falling transition. Power Compiler supports a library model that enables you to designate a separate rising and falling power value, depending on the transition.

Switching Power Calculation

Power Compiler analysis calculates switching power (P_c) in the following way:

$$P_c = \frac{V_{dd}^2}{2} \sum_{\forall \text{nets}(i)} (C_{Load_i} \times TR_i)$$

Where:

P_c Switching power of the design
 TR_i Toggle rate of net i , transitions per second
 V_{dd} Supply voltage

C_{Load_i} is the total capacitive load of net i , including parasitic capacitance, gate capacitance, and drain capacitance of all the pins connected to the net i .

Power Compiler software obtains C_{Load_i} from the wire load model for the net and from the logic library information for the gates connected to the net. You can also back-annotate capacitance information after physical design.

Dynamic Power Calculation

Because dynamic power is the power dissipated when a circuit is active, the equations for switching power and internal power provide the dynamic power of the design.

Dynamic power = Switching power + Internal power

For more information about the library models, see the *Library Compiler User Guide*.

Dynamic Power Unit Derivation

The unit for switching power and the values in the `internal_power` table is a derived unit. It is derived from the following function:

$(\text{capacitive_load_unit} * \text{voltage_unit}^2) / \text{time_unit}$

The function's parameters are defined in the library. The result is scaled to the closest MKS unit: micro, nano, femto, or pico. This dynamic power unit scaling effect needs to be taken into account by library developers when generating energy values for the internal power table.

The following is an example of how Power Compiler derives dynamic power units (if the library has the following attributes):

```
capacitive_load_unit (0.35, ff);
voltage_unit: "1V"
time_unit: "1ns";
```

To obtain the dynamic power unit, complete the following steps:

1. Find the starting value.

```
starting_value = capacitive_load_unit*voltage_unit2/
time_unit
starting_value = .35e-15*(1^2)/1e-9
starting_value = 3.5e-7W
```

The starting value consists of a base unit (1e-7W) and a multiplier (3.5).

2. Select an MKS base unit that converts the multiplier of the starting value found in step 1 to an integer number. For example, select an MKS unit between the range of att [1e-18] and giga [1e+12] watts, which converts the starting value's multiplier into an integer value.

The MKS base unit that meets this requirement in this example is nano [1e-9]. This is because the starting value of 3.5e-7W expressed in nW becomes 350nW. The original multiplier of 3.5 is converted to an integer value (350) by selecting the nW MKS base unit.

```
converted_value = 350e-9W
converted_value_multiplier = 350
base_unit = 1e9W = 1nW
```

3. Determine the base unit multiplier by selecting a power of 10 integer (for example, 1, 10, 100, ...) closest in magnitude to the converted value multiplier found in step 2.

```
converted_value_multiplier = 350 (from step 2)
base_unit_multiplier = 100
```

4. Combine the base unit multiplier obtained in step 3 and the base unit obtained in step 2 to obtain the dynamic power unit.

```
base_unit = 1nW (from step 2)
base_unit_multiplier = 100 (from step 3)
dynamic_power_unit = (100) 1nW = 100nW
```

In this example, each cell's dynamic power calculated by Power Compiler is multiplied by 100nW.

Power Calculation for Multirail Cells

Power Compiler supports the power analysis of libraries which contain cells with multiple rails for which power values are defined per voltage rail.

For multivoltage cells which contain separate power tables for each power level, Power Compiler determines the internal and leakage power contribution for each power rail and sums it to report the total power consumption.

For more information about defining per-rail power tables, see the *Library Compiler User Guide*.

The following example shows example cells that contain power tables per rail.

```
cell (AND2_1) {
  area : 1.0000;
  cell_footprint : MV12AND2;
  rail_connection (PV1, VDD1);
  rail_connection (PV2, VDD2);

  pin (a) {
    direction : input;
    capacitance : 0.1;
    input_signal_level : VDD1;
    internal_power () {
      power_level : VDD1;
      power (scalar) { values ( "1.0" ); }
    }
  }
  pin (b) {
    direction : input;
    capacitance : 0.1;
    input_signal_level : VDD1;
    internal_power () {
      power_level : VDD1;
      power (scalar) { values ( "1.0" ); }
    }
  }
  pin (y) {
    direction : output;
    function : "a & b";
    output_signal_level : VDD2;

    timing () {
      related_pin : "a";
      timing_sense : positive_unate;
      cell_rise ( scalar ) { values ( "1.0" ); }
      rise_transition ( scalar ) { values ( "1.0" ); }
      cell_fall ( scalar ) { values ( "1.0" ); }
      fall_transition ( scalar ) { values ( "1.0" ); }
    }
    timing () {
      related_pin : "b";
      cell_rise ( scalar ) { values ( "1.0" ); }
      rise_transition ( scalar ) { values ( "1.0" ); }
      cell_fall ( scalar ) { values ( "1.0" ); }
      fall_transition ( scalar ) { values ( "1.0" ); }
    }
    internal_power () {
      power_level : VDD1;
      power (scalar) { values ( "1.0" ); }
    }
    internal_power () {
      power_level : VDD2;
    }
  }
}
```

```

        power (scalar) { values ( "2.0" ); }
    }
}
leakage_power () {
    power_level : VDD1;
    value : 1.0;
}
leakage_power () {
    power_level : VDD2;
    value : 2.0;
}
cell_leakage_power : 10;
}

```

Using CCS Power Libraries

CCS power libraries contain unified library data for power and rail analysis and optimization, which ensures consistent analysis and simplification of the analysis flow. By capturing current waveforms in the library, you can provide more accurate identification of potential problem areas.

Both CCS and NLPM data can coexist in a cell description in the .lib file. That is, a cell description can have only NLPM data, only CCS data, or both NLPM and CCS data. Power Compiler uses either NLPM data or CCS data for the power calculation.

Use the `power_model_preference nlp | ccs` variable to specify your power model preference when the library contains both NLPM and CCS in it. The default is `nlp`. Using CCS or NLPM power libraries does not change the use model.

For more information about CCS power libraries and how to generate them, see the *Library Compiler User Guide*.

Voltage Scaling

Power Compiler uses the scaling library groups to implement temperature and voltage scaling. For voltage scaling, the libraries in the scaling group must contain CCS and NLPM power models.

To enable the scaling feature and specify the membership of libraries to scaling library groups, use the `define_scaling_lib_group` command. You can specify different scaling library groups for different design objects or subdesigns by using the `set_scaling_lib_group` command. To create an intermediate operating condition, use the `create_operating_conditions` command. Use the `set_operating_conditions` command to set intermediate voltage or temperature conditions. With the specified

operating conditions, the tool performs interpolation between the libraries in the library groups to obtain accurate delay information.

For more information about defining and setting the scaling library groups, see the related command man pages.

To perform both voltage and temperature scaling at the same time, use four libraries in the scaling group rather than two, representing the four possible combinations of voltage and temperature extremes: high voltage and high temperature, high voltage and low temperature, low voltage and high temperature, and low voltage and low temperature.

Power Compiler supports power scaling on both single-rail and multirail cells.

Script Examples for Voltage Scaling

[Example 3-1](#) shows an example script to perform voltage scaling in a multivoltage design.

Example 3-1 Voltage Scaling in Multivoltage Designs

```
read_verilog rtl.v
current_design top
link
define_scaling_lib_group -name group1{slow_0p81v.db slow_1p2v.db}
set_scaling_lib_group -min group1 -max group1
create_operating_conditions -name scaled_pvt -library slow_0p81.v \
    -process 1 -voltage 1.0 -temperature -40
set_operating_conditions -help
```

[Example 3-2](#) shows an example script to perform voltage scaling in a multicornner-multimode design.

Example 3-2 Voltage Scaling in Multicornner-Multimode Designs

```
read_verilog rtl.v
current_design top
link
define_scaling_lib_group -name group1{slow_0p81v.db slow_1p2v.db}
set_scaling_lib_group -min group1 -max group1
create_operating_conditions -name scaled_pvt -library slow_0p81.v \
    -process 1 -voltage 1.0 -temperature -40

create_scenario s1
read_sdc s1.sdc
set_operating_conditions scaled_pvt
set_tlu_plus_files -max_tluplus tlu_file1 -tech2itf_map map_file1
set_scaling_lib_group {group1}
```


Part 2: Power Analysis

4

Generating SAIF Files

Power Compiler requires information about the switching activity of your design to perform power analysis and power optimization. You can use simulation tools such as VCS to generate switching activity information for your design in Switching Activity Interchange Format (SAIF). This chapter describes how to generate switching activity in SAIF.

This chapter contains the following sections:

- [About Switching Activity](#)
- [Introduction to SAIF Files](#)
- [Generating SAIF Files](#)
- [Verilog Switching Activity Examples](#)
- [VHDL Switching Activity Example](#)

About Switching Activity

The dynamic power component usually accounts for a large percentage of the total power consumption in a combinational circuit. Dynamic power is the sum of the internal power of cells and the switching power. Switching power is the rate of energy usage resulting from the charging and discharging of capacitive loads during transitions between the two logic states, 0 and 1. Switching power depends on the clock rate and also the rate at which toggling occurs between logic states on each net. The toggle rate depends on the data being processed during typical usage of the logic circuit.

Power Compiler models switching activity based on the following:

- **Static Probability**

The fraction of time that a signal is at the logic 1 state. For example, a static probability of 0.8 means that the signal is in the logic 1 state 80 percent of the time and the logic 0 state 20 percent of the time.

- **Toggle Rate**

The rate at which a signal changes from 0 to 1 and from 1 to 0, in number of transitions per time unit.

When the switching activity information is available, you should annotate this information on the design objects so the tool can use the switching activity information during power optimization and analysis.

For more information about annotating switching activity, see [Annotating Switching Activity](#).

Introduction to SAIF Files

The accuracy of power calculations depends on the accuracy of the switching activity data. This data is generated using RTL simulation or gate-level simulation and is stored in a SAIF file. You should use the SAIF file to annotate switching activity information on the design objects before you perform power optimization and analysis.

SAIF is an ASCII format supported by Synopsys to facilitate the interchange of information between various Synopsys tools (see the *IEEE 1801 Standard, Annex J*). Use the `read_saif` command to read the SAIF file and the `write_saif` command to write out the SAIF file.

For more information, see the `read_saif` and the `write_saif` command man pages.

Early in the design cycle, you can use RTL simulation to determine the high-level switching and power characteristics of the design. Later in the design cycle, you can use gate-level

simulation to get more detailed switching data to annotate your design. The detailed switching data increases the accuracy of the power optimization and power analysis.

[Table 4-1](#) summarizes the various methods of generating SAIF files and their accuracies.

Table 4-1 Comparing Methods of Capturing Switching Activity

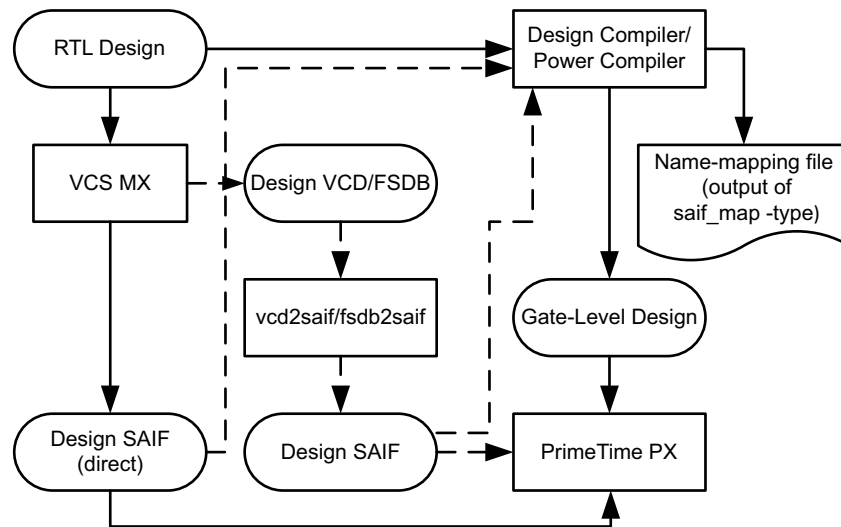
Simulation	Captured	Not captured	Trade-offs
RTL	Synthesis-invariant elements	1. Internal nodes 2. Correlation of non-synthesis-invariant elements 3. Glitching 4. State and path dependencies	Fast runtime at expense of some accuracy
Zero-delay and unit-delay gate-level	1. Synthesis-invariant elements 2. Internal nodes 3. Correlation 4. State dependencies 5. Some path dependencies	1. Some path dependencies 2. Glitching	More accurate than RTL simulation, but significantly higher runtime
Full-timing gate-level	1. All elements of design 2. Correlation 3. State and path dependencies	Highest accuracy, but runtime can be very long	Correlation between primary inputs

Generating SAIF Files

You can generate a SAIF file either from RTL simulation or gate-level simulation. This section discusses both RTL and gate-level simulation using Synopsys VCS. VCS supports Verilog, SystemVerilog, and VHDL formats.

Figure 4-1 shows two ways of generating a SAIF file. The solid lines indicate the suggested SAIF flow while the dotted lines indicate the alternative method of SAIF flow using various Synopsys tools.

Figure 4-1 SAIF File Generation and its Usage With Various Synopsys Tools



The following sections describe ways of generating SAIF files:

- [Generating SAIF Files From Simulation](#)
- [Generating SAIF Files From VCD Files](#)
- [Generating SAIF Files from FSDB Output Files](#)

You can read the SAIF file into Power Compiler and generate a mapping file for all the name changes of the nodes. You then read the name-mapping file and the synthesized gate-level netlist in PrimeTime PX to perform averaged power analysis

Generating SAIF Files From Simulation

VCS MX can generate the SAIF file directly from simulation. This direct SAIF file is smaller than VCD or FSDB files. Your input design for simulation can be an RTL or gate-level design. The design can be in Verilog, SystemVerilog, VHDL, or mixed HDL formats. When your design is in Verilog or SystemVerilog formats, you must specify system tasks to VCS MX using toggle commands. If your design is in VHDL format, use the power command as described in [Generating SAIF Files From VHDL Simulation](#).

For more information about the various supported formats and mixed language formats, see the *VCS MX User Guide*.

When generating the SAIF file during simulation, use the default monitoring policy (see the *VCS MX User Guide* for more information). This monitoring captures the switching activity of only the synthesis-invariant objects such as ports, tristate cells, black box cells, flip-flops, latches, retention registers, and hierarchical cells other than clock-gating cells. Integrated clock-gating cells and latch-based isolation cells are synthesis-dependent objects and therefore not captured.

If the library forward SAIF file contains details of state and path dependencies, the backward SAIF file generated also contains these details. For more information, see [Capturing State- and Path-Dependent Switching Activity](#).

The steps to generate SAIF files from simulation are discussed in the following sections:

- [Generating SAIF Files From SystemVerilog or Verilog Simulations](#)
- [Generating SAIF Files From VHDL Simulation](#)

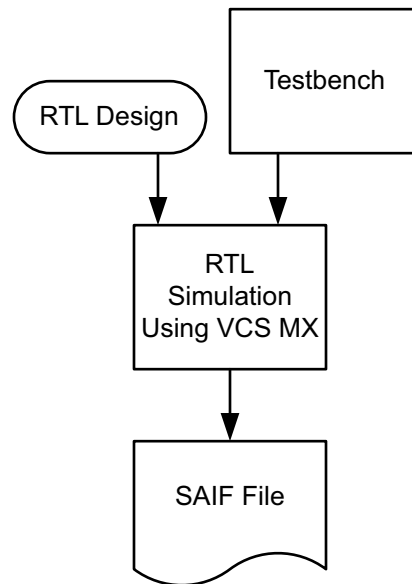
Generating SAIF Files From SystemVerilog or Verilog Simulations

Using VCS MX, you can generate SAIF files from both RTL and gate-level Verilog designs. When your design is in Verilog format, you must specify system tasks to VCS MX. These system tasks are also known as toggle commands. The system tasks specify the module for which switching activity is to be recorded and reported in the SAIF file. They also control the toggle monitoring during simulation.

For details about the toggle commands, see [VCS MX Toggle Commands](#).

Generating SAIF Files From RTL Simulation

[Figure 4-2](#) presents the methodology to capture switching activity using RTL simulation. RTL simulation captures the switching activity of primary inputs, primary outputs, and other synthesis-invariant elements.

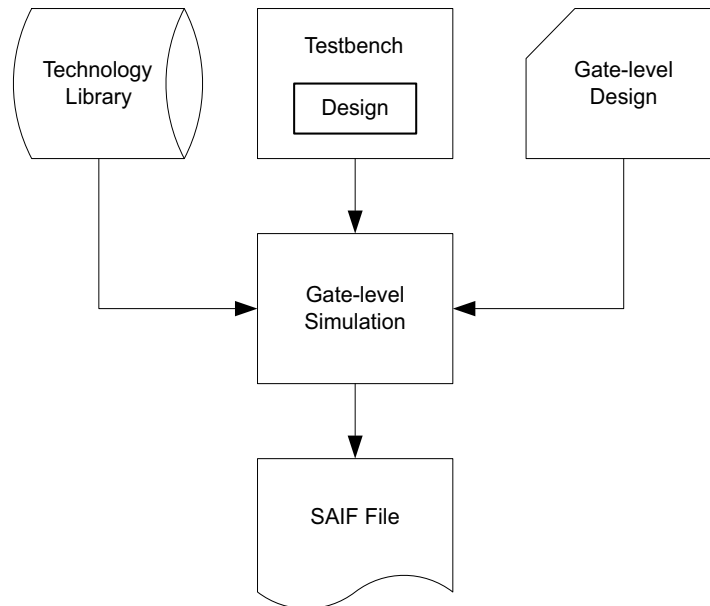
Figure 4-2 RTL Simulation Using VCS MX

To capture the switching activity using RTL simulation, specify the appropriate testbench and run the simulation.

The SAIF file contains the switching activity information of the synthesis-invariant elements in your design. To use the SAIF file for synthesis in the Power Compiler tool, annotate the switching activity, as described in [Annotating Switching Activity](#).

Generating SAIF Files From Gate-Level Simulation

[Figure 4-3](#) presents the methodology to capture switching activity using gate-level simulation. Gate-level simulation captures switching activity of pins, ports, and nets in your design.

Figure 4-3 Gate-Level Simulation Using VCS MX

To capture switching activity using gate-level simulation, specify the appropriate toggle commands in the testbench and run the simulation.

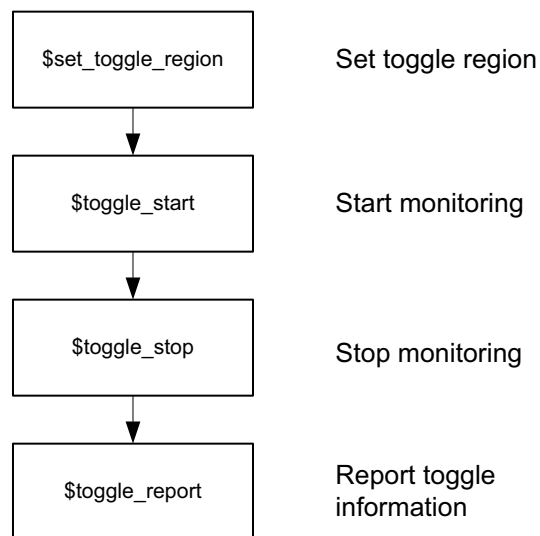
The SAIF file contains information about the switching activity of the pins, ports, and nets in your design. It can represent the pin-switching activity, based on rise and fall values, if your logic library has separate rise and fall power tables.

To use the SAIF file for synthesis in the Power Compiler tool, annotate the switching activity as described in [Annotating Switching Activity](#).

VCS MX Toggle Commands

To generate the SAIF file from RTL or gate-level Verilog or SystemVerilog, use toggle commands to specify system tasks to VCS MX. Using the toggle commands, you can specify the subblock for toggle counting and define specific periods for toggle counting during simulation. You can also control the start and stop of toggle counting.

[Figure 4-4](#) presents an overview of the toggle commands in your testbench file. Each toggle command starts with the \$ symbol. For simplicity, the figure does not show optional commands.

Figure 4-4 Toggle Command Flow

The system tasks that you specify to VCS MX using the toggle commands are

1. Define the toggle region.

The `$set_toggle_region` command specifies the module instance for which the simulator records the switching activity in the generated SAIF file. The syntax of this command is as follows:

```
$set_toggle_region(instance [, instance]);
```

When you explicitly mention one or more module instances as the toggle region, the simulator registers these objects and monitors them during simulation.

Note:

For gate-level simulation, if the logic library cell pins have rise and fall power values, their switching activity is monitored and reported for rise and fall separately.

2. Begin toggle monitoring.

Use the `$toggle_start` command to instruct the simulator to start monitoring the switching activity. The syntax of this command is as follows:

```
$toggle_start();
```

During simulation, the tool starts monitoring the switching activity of the module instances that are defined in the toggle region.

3. End toggle monitoring.

Use the `$toggle_stop` command to instruct the simulator to stop monitoring the switching activity.

4. Report toggle information in an output file.

Use the `$toggle_report` command to write monitored gate and net switching activity to an output file. You can invoke `$toggle_report` any number of times using different parameters. For more details and examples of SAIF files, see [RTL SAIF File](#).

The syntax for the `$toggle_report` command is as follows:

```
$toggle_report (filename, [synthesis_time_unit], instance_name_string);
```

The values for the various options and parameters are as follows:

- *filename*

This is the name of the switching activity output file.

- *synthesis_time_unit*

This optional parameter is the time unit of your synthesis library, in seconds. For example, if the time unit in your synthesis library is 10 picoseconds, specify 1.0e-11.

The `$toggle_report` command uses this number to convert simulation time units to synthesis time units. Power Compiler obtains the simulation time unit from simulation. If you don't specify the synthesis time unit parameter, the default is 1 ns (1.0e-9).

- *instance_name_string*

This required parameter is the full instance path name of the block from the top of your simulation environment down to the name of the block instance to be reported.

Example

```
$toggle_report ("file.saif", 1.0e-11, "test.DUT");
```

In this example, the file written out is file.saif, the synthesis time unit is 10 picoseconds, and the name of the monitored instance is test.DUT. The output file format is SAIF, which is the default.

Resetting the Toggle Counter

Use the `$toggle_reset` command to set the toggle counter to 0 for all the nets in the current toggle region. This command starts a new toggle monitoring period in a simulation session.

For example, when using `$toggle_start`, `$toggle_stop` or `$toggle_reset` with `$toggle_report`, you can create SAIF output files for specific periods during simulation. The syntax of this command is as follows:

```
$toggle_reset();
```

Use the `$toggle_reset` command only after you have written out the previous results with the `$toggle_report` command.

Capturing State- and Path-Dependent Switching Activity

By default, Power Compiler estimates the state- and path-dependent power information that is required for power calculations. However, if you want to obtain this information through simulation, you can use the `lib2saif` command before simulation. In this case, given a logic library, you can run the utility to obtain a library SAIF file that contains the directives for generating state- and path-dependent switching information. This file is called the library forward SAIF file. This file becomes an input to gate-level simulation.

The library forward SAIF file contains information from the logic library about cells that have state and path dependencies. It can have rise and fall information if the library has separate rise and fall power tables.

To read the library forward SAIF file into the simulator, use the `$read_lib_saif` command. This command registers the state- and path-dependent information for monitoring during simulation.

The syntax of the `$read_lib_saif` command is as follows:

```
$read_lib_saif(input_file);
```

For gate-level simulation, you must use the `$read_lib_saif` command to register state- and path-dependent cells and, by default, all internal nets in the design. The command registers state-dependent and path-dependent cells by reading the library forward SAIF file. In addition, you must also set the toggle region for monitoring. If you do not use the `$read_lib_saif` command, the simulator registers all internal nets for monitoring by default.

You can use the `$read_lib_saif` command as often as you require during simulation; however, you must use this command before defining the toggle region using the `$set_toggle_region` command. When you define the toggle region, the `$set_toggle_region` command checks for the presence or absence of a previous `$read_lib_saif` command and registers internal nets accordingly.

[Example 4-1](#) shows an example of a SAIF file generated by the `lib2saif` command.

Example 4-1 File Generated by `lib2saif`

```
(SAIFILE
(SAIFVERSION "2.0" "lib")
(DIRECTION "forward")
(DESIGN )
(DATE ...)
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "Power Compiler lib2saif") (VERSION ...) (DIVIDER / )
(LIBRARY "example"
  (MODULE "AND2"
    (PORT
      (Z
        (IOPATH A IOPATH B)
      )
    )
  )
)
```



```

    )
  )
  (MODULE "DFF1"
    (LEAKAGE
      (COND Q
        COND !Q
        COND_DEFAULT)
    )
  )
  (MODULE "EXOR3"
    (PORT
      (Z
        (COND ((!B * !A) | (B * A)) RISE_FALL (IOPATH C)
          COND ((!B * A) | (B * !A)) RISE_FALL (IOPATH C)
          COND ((!C * !A) | (C * A)) RISE_FALL (IOPATH B)
          COND ((!C * A) | (C * !A)) RISE_FALL (IOPATH B)
          COND ((!C * !B) | (C * B)) RISE_FALL (IOPATH A)
          COND ((!C * B) | (C * !B)) RISE_FALL (IOPATH A)
          COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH C))
        )
      )
    )
  (MODULE "MUX21"
    (PORT
      (Z
        (COND (B * !A) RISE_FALL (IOPATH S)
          COND (!B * A) RISE_FALL (IOPATH S)
          COND_DEFAULT RISE_FALL (IOPATH A IOPATH B IOPATH S))
        )
      )
    (LEAKAGE
      (COND (B * S * A)
        COND (!B * S * A)
        COND (!B * !S * A)
        COND (!A * S * B)
        COND (!A * !S * B)
        COND_DEFAULT)
    )
  )
  (MODULE "NAND2"
    (PORT
      (Z
        (IOPATH A IOPATH B)
      )
    )
  )
  (MODULE "OR2"
    (PORT
      (Z
        (IOPATH A IOPATH B)
      )
    )
  )

```

```

. . . .

(MODULE "iopad6"
  (PORT
    (PAD
      (COND !TS RISE_FALL (IOPATH DI)
        COND_DEFAULT RISE_FALL)
    )
    (DI
      (COND TS RISE_FALL
        COND_DEFAULT RISE_FALL)
    )
    (DO
      (IOPATH PAD IOPATH_DEFAULT)
    )
  )
)
)

```

Overriding Default Registration of Internal Nets

After you have run the `read_lib_saif` command in the testbench, you can override the default net monitoring behavior using the `$set_gate_level_monitoring` command. This command turns on or turns off the registration of internal nets.

The following is the syntax of the `$set_gate_level_monitoring` command:

```
$set_gate_level_monitoring ("on" | "rtl_on", "mda" | "sv");
```

"on"

This string explicitly registers all internal nets for simulation. Thus, simulation monitors any internal net in the region defined by the `$set_toggle_region` command.

"rtl_on"

The registers in the toggle region are monitored and the nets in the toggle region are not monitored during simulation.

"mda"

Use this argument for Verilog memories and multidimensional arrays.

"sv"

Use this argument for SystemVerilog data objects.

The `$set_gate_level_monitoring` command is optional. If you use it, you must do so before invoking the `$set_toggle_region` command.

Generating SAIF Files From VHDL Simulation

You can use VCS MX to generate SAIF files from RTL or gate-level simulation of VHDL designs. The methodology to generate the SAIF file is similar to the methodology used for Verilog designs, shown in [Figure 4-2](#) and [Figure 4-3](#). However, you cannot use the toggle commands to specify the system tasks to the simulator.

For RTL-level VHDL files, variables are not supported by the simulator for monitoring. However, VHDL constructs such as generates, enumerated types, records, and arrays of arrays are supported by VCS MX for simulation.

The use model to generate a SAIF file from VHDL simulation consists of using the power command at the VCS MX command line interface, simv. The syntax of the power command is as follows:

```
power
  -enable
  -disable
  -reset
  -report file_name synthesis_time_unit scope
  -rtl_saif file_name
  [test_bench_path_name]
  -gate_level on| off | rtl_on
  region_signal_variable
```

- The -enable option enables the monitoring of the switching activity.
- The -disable option disables the monitoring of the switching activity.
- The -reset option resets the toggle counter.
- The -report option reports the switching activity to an output SAIF file.
- The -rtl_saif option reads the RTL forward-SAIF file.
- You can use on, off, or rtl_on with the -gate_level option. [Table 4-2](#) summarizes the monitoring policy for VHDL simulation.

Table 4-2 Monitoring Policy for VHDL Simulation

Monitoring policy	Ports	Signals	Variables
on	Yes	Yes	No
off	No	No	No
rtl_on	Yes	Yes	No

- You can specify either the hierarchical path to the signal name or the toggle region and its children to be considered for monitoring.

System Task List for SAIF File Generation From VHDL Simulation

The following example script shows a task list that you specify to the simulator to generate a SAIF file. The design name in the example is test1. You can either specify each of these commands at the VCS MX command prompt or run the file that contains these commands.

```
power test1
power -enable
run 10000
power -disable
power -report vhdl.saif 1e-09 test
quit
```

Generating SAIF Files From VCD Files

You can generate SAIF files from VCD files. To generate a SAIF from a VCD file generated by the VCS tool, use the `vcd2saif` utility. Follow these steps to generate the SAIF file and to annotate the switching activity:

1. Run the simulation to generate VCD file.
2. Use the `vcd2saif` utility to convert the VCD file to a SAIF file.
3. Annotate the switching activity within the SAIF file as described in [Annotating Switching Activity](#).

The disadvantage of using this method is that VCD files can be very large, especially for gate-level simulation, requiring more time for processing. Also, the SAIF file generated by the `vcd2saif` utility lacks state-dependent and path-dependent information.

Converting a VCD File to a SAIF File

The `vcd2saif` utility converts the RTL or gate-level VCD file generated by VCS into a SAIF file. This utility has limited capability when the VCD is generated from the SystemVerilog simulation as described in [Limited SystemVerilog Support in the vcd2saif Utility](#).

The `vcd2saif` utility is platform-specific and is located in `install_dir/$ARCH/syn/bin`. The `$ARCH` environment variable represents the specific platform (architecture) of your Synopsys software installation, such as linux or AMD.

You can use compressed VCD files (.Z) and gzipped VCD files (.gz). In addition, for VPD files, you can use the utility located at `$VCS_HOME/bin/vpd2vcd`, and for FSDB files, you can use the utility located at `$SYNOPSYS/bin/fsdb2vcd`.

The `vcd2saif` utility does not support state-dependent and path-dependent switching activity. For information about each option, use the `vcd2saif -help` command.

Limited SystemVerilog Support in the vcd2saif Utility

The `vcd2saif` utility supports only a limited set of SystemVerilog constructs for VCD files generated from SystemVerilog simulation. [Table 4-3](#) shows the list of SystemVerilog constructs that are supported by the `vcd2saif` utility.

Table 4-3 SystemVerilog Constructs Supported by the `vcd2saif` Utility

char	int	shortint	longint	bit	byte	logic
shortreal	void	enum	typedef	struct	union	arrays (packed and unpacked)

Generating SAIF Files from FSDB Output Files

There are two ways to generate a SAIF file from an FSDB file:

1. Using the `fsdb2saif` utility
2. Using the `fsdb2vcd` utility and then using the `vcd2saif` utility.

For more information about the FSDB utilities, see the *Verdi3 and Siloti Command Reference Manual*. After generating the SAIF file, annotate the switching activity from the SAIF file as described in [Annotating Switching Activity](#).

Verilog Switching Activity Examples

The following examples demonstrate RTL and gate-level descriptions with Verilog-generated switching activity data.

RTL Example

This Verilog RTL example includes the following elements:

- RTL design description
- RTL testbench
- SAIF output file from simulation

Verilog Design Description

[Example 4-2](#) shows the description for a state machine called test.

Example 4-2 RTL Verilog Design Description

```

`timescale 1 ns / 1 ns
module test ( data, clock, reset, dummy);

input [1:0] data;
input clock;
input reset;
output dummy;

wire dummy;

wire [1:0] NEXT_STATE;
reg [1:0] PRES_STATE;

parameter s0 = 2'b00;
parameter s5 = 2'b01;
parameter s10 = 2'b10;
parameter s15 = 2'b11;

function [2:0] fsm;
input [1:0] fsm_data;
input [1:0] fsm_PRES_STATE;

reg fsm_dummy;
reg [1:0] fsm_NEXT_STATE;

begin
case (fsm_PRES_STATE)
s0: //state = s0
begin
if (fsm_data == 2'b10)
begin
fsm_dummy = 1'b0;
fsm_NEXT_STATE = s10;
end
else if (fsm_data == 2'b01)
//....
end

s5: //state = s5
begin
// ...
end

s10: //state = s10
begin
// ...
end

s15: //state 15
begin

```

```
        // ...
    end
endcase

    fsm = {fsm_dummy, fsm_NEXT_STATE};
end

endfunction

assign {dummy, NEXT_STATE} = fsm(data, PRES_STATE);

always @(posedge clock)
begin
    if (reset == 1'b1)
    begin
        PRES_STATE = s0;
    end
    else
    begin
        PRES_STATE= NEXT_STATE;
    end
end
endmodule
```

RTL Testbench

The Verilog testbench in [Example 4-3](#) simulates the design test described in [Example 4-2](#). The testbench instantiates the design test as U1.

Example 4-3 RTL Testbench

```

`timescale 1 ns / 1 ns

module stimulus;

reg clock;
reg [1:0] data;
reg reset;
wire dummy;
test U1 (data,clock, reset, dummy);

always
begin
    #10 clock = ~clock;
end

initial
begin
    $set_toggle_region(stimulus.U1);
    $toggle_start();
    clock = 1'b0;
    data = 2'b00;
    reset = 1'b1;
    #50 reset = 0;
    #25 data = 3; #20 data = 0;
    #20 data = 1; #20 data = 2;
    // ...
    $toggle_stop();
    $toggle_report("my_rtl_saif", 1.0e-12, "stimulus");
    #80 $finish;
end

```

RTL SAIF File

The RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `$toggle_report` command creates this file.

[Example 4-4](#) is a SAIF file created for the RTL Verilog description that is also shown in [Example 4-2](#) and for the testbench shown in [Example 4-3](#).

Example 4-4 RTL SAIF File

```

/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets **/
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(
  DATE "Fri Feb 6 14:21:20 2015"
  (VENDOR "Synopsys, Inc")

```



```

(PROGRAM_NAME "VCS I-2014.03-SP1")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ps)
(DURATION 135000.00)
(INSTANCE stimulus
  (INSTANCE U1
    (NET
      (data\[1\]
        (T0 115000) (T1 20000) (TX 0)
        (TC 2) (IG 0)
      )
      (data\[0\]
        (T0 95000) (T1 40000) (TX 0)
        (TC 3) (IG 0)
      )
      (clock
        (T0 70000) (T1 65000) (TX 0)
        (TC 13) (IG 0)
      )
      (reset
        (T0 85000) (T1 50000) (TX 0)
        (TC 1) (IG 0)
      )
      (dummy
        (T0 0) (T1 0) (TX 135000)
        (TC 0) (IG 0)
      )
      (NEXT_STATE\[1\]
        (T0 0) (T1 0) (TX 135000)
        (TC 0) (IG 0)
      )
      (NEXT_STATE\[0\]
        (T0 0) (T1 0) (TX 135000)
        (TC 0) (IG 0)
      )
    )
  )
)
)
)

```

Understanding the SAIF File

[Table 4-4](#) summarizes the definitions for various SAIF file terminology:

Table 4-4 Definitions of SAIF File Terminology

T0	Duration of time found in logic 0 state.
T1	Duration of time found in logic 1 state.
TX	Duration of time found in unknown "X" state.

Table 4-4 Definitions of SAIF File Terminology (Continued)

TC	The sum of the rise (0-to-1) and fall (1-to-0) transitions that are captured during monitoring.
IG	Number of 0 - X - 0 and 1 - X - 1 glitches captured during monitoring.
RISE	Rise transitions in a given state.
FALL	Fall transitions in a given state.

Duration refers to the time span between \$toggle_start and \$toggle_stop commands in the testbench during simulation. During this time span, ports, pins, and nets are monitored for toggle activity. For more information on the terminology of the SAIF file, see the *IEEE 1801 Standard, Annex J*.

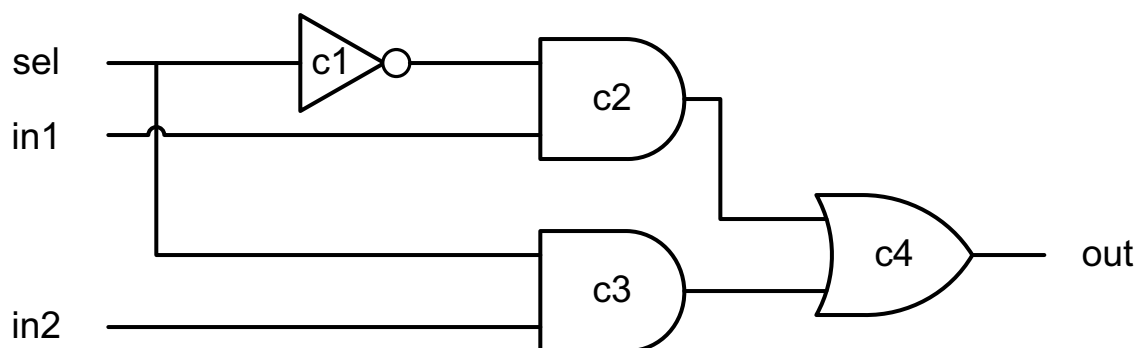
Gate-Level Example

This Verilog gate-level example illustrates the following elements:

- Verilog cell description and schematic
- Verilog testbench
- SAIF output file from simulation

Gate-Level Verilog Module

Figure 4-5 shows the schematic for a simple multiplexer.

Figure 4-5 Schematic of Multiplexer Circuit: MUX21

[Example 4-5](#) is the Verilog module that describes the MUX21 design.

Example 4-5 Verilog Module of Multiplexer Circuit: MUX21

```
/*'timescale 10ps/ 1ps
*/
module MUX21(out,d1,d2,sel);
input d1, d2, sel;
output out;
    IV c1(.Z(sel_),.A(sel));
    AN2 c2(.Z(d1m),.A(d1),.B(sel_));
    AN2 c3(.Z(d2m),.A(d2),.B(sel));
    OR2 c4(.Z(out),.A(d1m),.B(d2m));
endmodule
```

Verilog Testbench

The Verilog testbench in [Example 4-6](#) tests the MUX21 design by simulating it and monitoring the various signals.

Example 4-6 Verilog Testbench for MUX21

```
/* Begin test.v */
'timescale 1ns/ 10ps
module top;
    reg in1, in2, sel;
    parameter hazrate = 0.99;
    parameter haztime = 0.23;

    MUX21 m1(out,in1,in2,sel);

    initial
    begin
        // start monitoring
        $monitor($time,,,"in1=%b in2=%b sel=%b
        out=%b",in1,in2,sel,out);

        // read SAIF file of state or path dependent information
        $read_lib_saif (cell.saif);

        // define the monitoring scope
        $set_toggle_region (m1);

        $toggle_start;

        // test first data line passing 0
        sel = 0;
        in1 = 0;
        in2 = 0;

        // test first data line passing 1
        #10 in1 = 1;
```

```

        #10 sel = 1;

        // test second data line passing 1
        #10 in2 = 1;

        $toggle_stop;
        $toggle_report("my_1st", 1.0e-9, "top.m1", hazrate, haztime);

        // exit simulation
        $finish(2);
    end
endmodule

```

The `$set_toggle_region` command sets the monitoring scope in module `m1` (the testbench instantiation of `MUX21`). All subsequent toggle commands affect only registered design objects and designs instantiated in registered objects. Thus, under `m1`, simulation monitors internal nets and state- and path-dependent cells (in this simple example, however, there are no subdesigns in `m1`).

The testbench example invokes `$toggle_report` command before exiting the simulation. Make sure that you declare any parameters you use for `$toggle_report` command in your testbench. These parameters appear at the top of the testbench in [Example 4-6](#).

Gate-Level SAIF File

[Example 4-7](#) shows a SAIF file generated from gate-level simulation of `MUX21`.

Example 4-7 \$toggle_report Output File in SAIF

```

(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Fri Oct 6 18:58:58 2000")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-MX Power Compiler")
(VERSION "3.3")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 99999.00)
(INSTANCE tb
  (INSTANCE dut
    (NET
      (n12159
        (T0 99529) (T1 470) (TX 1)
        (TC 46) (IG 0)
      )
      (n12480
        (T0 0) (T1 99998) (TX 0)
        (TC 0) (IG 0)
      )
    )
  )
)

```

```
)
(n12117
    (T0 61) (T1 99938) (TX 0)
    (TC 26) (IG 0)
)
)
(INSTANCE U12053
    (PORT
        (Z
            (T0 10) (T1 99989) (TX 0)
            (COND A (RISE)
                (IOPATH B (TC 0) (IG 0)
            )
            COND A (FALL)
                (IOPATH B (TC 0) (IG 0)
            )
            COND B (RISE)
                (IOPATH A (TC 0) (IG 0)
            )
            COND B (FALL)
                (IOPATH A (TC 1) (IG 0)
            )
            COND_DEFAULT (TC 1) (IG 0)
        )
    )
)
)
```

VHDL Switching Activity Example

This VHDL RTL example includes the following elements:

- RTL design description
- RTL testbench
- SAIF output file from simulation

VHDL Design Description

Example 4-8 shows the description for a design called dummy.

Example 4-8 RTL VHDL Design Description

```
library ieee;
use ieee.std_logic_1164.all;
entity dummy is
```

```
architecture beh of dummy is
  signal clk: std_logic := '0';
begin
  clk <= not clk after 5 ns;
end beh;
```

RTL Testbench

The RTL testbench in [Example 4-9](#) simulates the design test described in [Example 4-8](#). The testbench instantiates the design dummy as dummy_ins.

Example 4-9 RTL Testbench

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
end entity
architecture testbench of test is
  component dummy is
  end component;
begin
  dummy_ins: dummy;
end testbench;
```

RTL SAIF File

This RTL SAIF file is the output of RTL simulation and contains information about the switching activity of synthesis-invariant elements. The `power -report` command creates this file.

[Example 4-10](#) is a SAIF file for the RTL VHDL description that is shown in [Example 4-8](#).

Example 4-10 RTL SAIF File

```

/** There is no explicit set_gate_level_monitoring command, **/
/** and the default behavior is to monitor internal nets **/
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DSIGN )
(DATE "Tue May 5 05:56:35 2009")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-Scirocco-MX Power Compiler")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 10000.00)
(INSTANCE TEST
  (INSTANCE DUMMY_INS
    (NET
      (CLK
        (T0 5000) (T1 5000) (TX 0)
        (TC 1999) (IG 0)
      )
    )
  )
)
)
)

```


5

Annotating Switching Activity

Switching activity is required for accurate power calculations. This chapter explains the different types of switching activity information and illustrates how you can annotate switching activity on gate-level design objects.

This chapter contains the following sections:

- [Types of Switching Activity to Annotate](#)
- [Annotating Switching Activity Using RTL SAIF Files](#)
- [Annotating Switching Activity Using Gate-Level SAIF Files](#)
- [Annotating Inferred Switching Activity](#)
- [Annotating Switching Activity Using the `set\_switching\_activity` Command](#)
- [Fully Versus Partially Annotating the Design](#)
- [Analyzing the Switching Activity Annotation](#)
- [Removing the Switching Activity Annotation](#)
- [Design Objects Without Annotated Switching Activity](#)

Types of Switching Activity to Annotate

The power of a design depends on the switching activity of the nets and cell pins. The switching activity is used by the `report_power` command during power calculation.

The following types of switching activity can be annotated on design objects:

- Simple switching activity on design nets, ports, and cell pins. Simple switching activity consists of the static probability and the toggle rate. The static probability is the fraction of the time that the object is at logic 1. The toggle rate is the rate at which the design object switches between logic 0 and logic 1.
- State-dependent toggle rates on input pins of leaf cells. As explained in [Power Modeling and Calculation](#), the internal power characterization of an input pin of a library cell can be state dependent. The input pins of instances of such cells can be annotated with state dependent toggle rates.
- State-dependent and path-dependent toggle rates on output pins of leaf cells. As explained in [Power Modeling and Calculation](#), the internal power characterization of output pins can be state dependent and path dependent. Output pins of cells with state- and path-dependent characterization can be annotated with state- and path-dependent toggle rates.
- State-dependent static probability on leaf cells. Cell leakage power can be characterized using state dependent leakage power tables (see [Power Modeling and Calculation](#)). Such cells can be annotated with state-dependent static probability.

Annotating Switching Activity Using RTL SAIF Files

Optimal power analysis and optimization results occur when switching activities reported in the RTL SAIF file are accurately associated with the correct design objects in the gate-level netlist. For this to occur, the RTL names must map correctly to their gate-level counterparts. During synthesis, however, mapping inaccuracies can occur that can affect your annotation.

To ensure proper name mapping and annotation for RTL SAIF files, do the following:

1. At the beginning of synthesis, specify the `saif_map -start` command.

This command causes Power Compiler to create a name-mapping database during synthesis optimization that Power Compiler then uses for power analysis and optimization.
2. Before compiling, specify the `read_saif -auto_map_names` command to perform RTL SAIF annotation using the name-mapping database.

Using the Name-Mapping Database

You can access the name-mapping database by using the `saif_map` command, which allows you to query, report, modify, save, clear, and load the database. If the object names require modification, use the `read_saif -auto_map_names` command. You can read a regular, uncompressed file or a compressed file in gzip format by using the `-input` option of the `saif_map` command. The `saif_map` command has the following syntax:

```
saif_map
  [-start]
  [-end]
  [-reset]
  [-report]
  [-get_name]
  [-set_name name_list]
  [-add_name name_list]
  [-remove_name name_list]
  [-clear_name]
  [-get_object_names name_list]
  [-create_map]
  [-write_map file_name]
  [-read_map file_name]
  [-type type]
  [-inverted]
  [-instances objects]
  [-no_hierarchical]
  [-columns columns]
  [-sort columns]
  [-rtl_summary]
  [-missing_rtl]
  [-input SAIF_file]
  [-review]
  [-preview]
  [-source_instance SAIF_instance_name]
  [-target_instance target_instance_name]
  [-hsep character]
  [-nosplit]
  [object_list]
```

After you run the `read_saif -auto_map_names` command, review the name-mapping database using the following commands:

```
read_saif -auto_map_names -input ../sim/rtl.saif \
  -instance_name tb/dut -verbose
report_saif -hier -rtl_saif -missing
```

You can manually add a mapping entry with the `saif_map -add_name` command as follows:

```
reset_switching_activity
saif_map -add_name "Ax_ins" [get_port AX_usr_ins]
read_saif -auto_map_names -input ../sim/rtl.saif \
  -instance_name tb/dut
```

In this example, you manually map the RTL SAIF object “Ax_ins” and the design object “AX_usr_ins.” When you run the `read_saif -auto_map_names` command, the Power Compiler tool performs annotation again using the modified database.

For more information about the command options, see the `read_saif` and `saif_map` command man pages.

Integrating the RTL Annotation With PrimeTime PX

Similar to Power Compiler, PrimeTime PX requires accurate RTL-to-gate name-mapping correspondence to perform accurate power analysis. Use Power Compiler to output the name-mapping files that PrimeTime PX can use for RTL-to-gate name mapping.

After using the `read_saif` command, specify the `saif_map` command as follows to generate a name-mapping file that can be read directly into PrimeTime PX:

```
saif_map -type ptpx -write_map file_name
```

The name-mapping output file appears as follows:

```
set_rtl_to_gate_name -rtl{clk_sn} -gate clk_sn
set_rtl_to_gate_name -rtl{rx_top/data_i[9]} \
    -gate rx_top_data_i_reg<9>
...
```

[Example 5-1](#) shows the recommended flow using RTL SAIF file to ensure optimal power analysis during synthesis and get the proper names for PrimeTime PX.

Example 5-1 Annotating Switching Activity Using RTL SAIF Files Flow

```
saif_map -start
read_verilog rtl_design.v
link
create_clock clk
read_saif -auto_map_names -input ../sim/rtl.saif \
    -instance_name tb/dut
report_saif -hierarchy -rtl_saif
compile_ultra
report_saif -hierarchy -rtl_saif
change_name -rules verilog -hierarchy
write -format verilog -hierarchy -output mapped_design.v
saif_map -type ptpx -write_map saifmap.ptpx.tcl
```

Annotating Switching Activity Using Gate-Level SAIF Files

You can use either the `read_saif` or `merge_saif` command to annotate switching activity. The `read_saif` command reads a SAIF file and annotates switching activity information on the nets, pins, and ports of the design.

The `merge_saif` command reads a list of SAIF files, computes the toggle rates and static probability, and annotates the switching activity information on the nets, pins, and ports of the design. This command creates a merged-output SAIF file.

Reading SAIF Files Using the `read_saif` Command

To annotate gate-level switching activity onto the gate-level netlist, use the `read_saif` command. For example,

```
dc_shell> read_saif -input myfile.saif -auto_map -instance_name T1/DUT/U1
```

In this example, the `read_saif` command annotates the information in the input file named `myfile.saif` onto the current gate-level design, `U1`. The `-instance_name` option identifies the hierarchical location of the current design in the simulation environment.

The input file specified using the `-input` option of the `read_saif` command can be a text file or a compressed gzip file with a `.gzip` extension. For example,

```
dc_shell> read_saif -input myfile.gzip -instance_name T1/DUT/U1
```

A SAIF file is usually generated in the HDL simulation flow, where a simulation testbench instantiates the design being simulated and provides simulation vectors. The generated SAIF file contains the switching activity information organized in a hierarchical fashion, where the hierarchy of the SAIF file reflects the hierarchy of the simulation testbench. If a design is instantiated in the testbench (tb) as the instance `i`, then the SAIF file contains the switching activity information for the design under the hierarchy `tb/i`. In this case, specify the `tb/i` instance name to the `-instance_name` option when reading the SAIF file as follows:

```
dc_shell> read_saif -input des.saif -instance_name tb/i
```

Specifying an invalid instance name results in having all or most of the switching activity stored in the SAIF file not read properly. An error message is printed if none of the information stored in the SAIF file is read by the `read_saif` command.

The SAIF file contains time duration values and specifies a time unit which is usually the time unit used during simulation. When reading the SAIF file, the `read_saif` command automatically converts the SAIF time units to the synthesis time units. The synthesis time units are obtained from the time units of the target or link library. When the synthesis time units cannot be obtained, the `read_saif` command prints a warning message and uses a default time unit of 1 ns. In such cases, the `-scale` and `-unit_base` options can be used to specify the intended synthesis time unit. For example, if a target library with the time units 100 ps is used for synthesis and a SAIF file is being read before the library is used (for linking or synthesis), use the options as follows:

```
dc_shell> read_saif ... -scale 100 -unit_base ps
```

When reading the SAIF file, the `report_lib` command gives the time units specified in a logic library. The `report_power` command gives the synthesis library time units used during power calculations.

The `read_saif` command has the following syntax:

```
read_saif
  -input file_name
  [-instance_name string]
  [-target_instance instance]
  [-names_file file_name]
  [-ignore string]
  [-ignore_absolute string]
  [-exclude file_name]
  [-exclude_absolute file_name]
  [-scale scale_value]
  [-unit_base time_unit]
  [-khrate float]
  [-map_names]
  [-auto_map_names]
  [-verbose]
```

For information about the command options, see the `read_saif` command man page.

Reading SAIF Files Using the `merge_saif` Command

The `merge_saif` command can be used to read switching activity information from multiple SAIF files. Input SAIF files are given individual weights, and a weighted sum of the switching activities is annotated. This command can be used in flows where different SAIF files are generated for different modes of the same design. The switching activity from all the different modes can then be used for power calculations and optimization.

The following example shows how to use the `merge_saif` command. In this example, the design has three modes: standby, slow, and fast; and the SAIF files are `standby.saif`, `slow.saif`, and `fast.saif`. Depending on the expected use of the design, specify the following weight for each SAIF file, with the total weight always equal to 100 percent:

standby.saif: 80%; slow.saif: 5%; fast.saif: 15%

The SAIF files are read as shown in the following example:

```
dc_shell> merge_saif -input_list \
    { -input standby.saif -weight 80 \
      -input slow.saif -weight 5 \
      -input fast.saif -weight 15 } \
  -instance_name tb/i
```

The `-output` option of the `merge_saif` command can be used to generate a SAIF file containing the weighted sum of the switching activities. When the output file is specified with

a .gzip extension, then a compressed file is written out in gzip format. If the output file is specified with a .saif extension, then a uncompressed SAIF format file is written.

After the `merge_saif` command reads each individual SAIF file, it uses a switching activity propagation mechanism to estimate the switching activity of design nets that are not included in the SAIF file. You can therefore use the following command to generate a gate-level SAIF file with estimated switching activity information from an RTL SAIF file:

```
dc_shell> merge_saif -input_list { -input rtl.saif -weight 100} \
                  -instance_name tb/i -output estimate.saif
```

The `-simple_merge` option can be used to switch off the switching activity propagation mechanism when the information in the SAIF files is being merged.

The syntax of the `merge_saif` command is the same as that of the `read_saif` command with the following exceptions:

- A weighted input file list is specified instead of a single input file
- The `-simple_merge` and `-output` options can be used with the `merge_saif` command

The `merge_saif` command has the following syntax:

```
merge_saif
  input_list weighted_filename_list
  [-simple_merge]
  [-output merged_saif_filename]
  [-instance_name string]
  [-scale scale_value]
  [-unit_base time_unit]
  [-ignore string]
  [-ignore_absolute string]
  [-exclude filename]
  [-exclude_absolute filename]
  [-map_names]
  [-khrate float]
```

For more information, see the `merge_saif` command man page.

Annotating Inferred Switching Activity

The `infer_switching_activity` command detects the drivers of special pins such as asynchronous set, asynchronous clear, synchronous set, and synchronous clear, and suggests values for toggle rate and static probability.

The `infer_switching_activity` command reports the current and proposed static probability and toggle rate, as shown in the following example:

```
dc_shell> infer_switching_activity
Information: Updating design information... (UID-85)

Created by infer_switching_activity ...
```

Objects	Type	Current Static Probability	Current Toggle Rate	Proposed Static Probability	Proposed Toggle Rate
U646/Z	driver	None	None	1.0	0.0
U645/ZN	driver	None	None	1.0	0.0

When you specify the `-apply` option, the proposed switching activity is annotated on the drivers listed in the output. After applying the switching activity, Power Compiler uses the applied values to report the power consumption.

For more information, see the `infer_switching_activity` command man page.

Annotating Switching Activity Using the `set_switching_activity` Command

The `set_switching_activity` command annotates switching activity on design objects such as pins, ports, nets, and cells. The types of activity that you can annotate include state- and path-dependent toggle rates and state-dependent static probabilities.

The `set_switching_activity` command has the following syntax:

```
set_switching_activity
  [-static_probability static_probability_value]
  [-toggle_rate toggle_rate]
  [-state_condition boolean_equation_of_pins]
  [-path_sources pins_of_the_source_of_this_path]
  [-rise_ratio rise_or_total_toggle_ratio]
  [-period period_value | -base_clock clock]
  [-type list_of_object_type]
  [-hierarchy]
  [-verbose]
  [object_list]
```

Use the `-static_probability` option to specify the static probability value, which is a floating-point number between 0.0 and 1.0. Static probability is the fraction of time that the signal is at logic 1.

Use the `-toggle_rate` option to specify the toggle rate value, which is a floating point number. Toggle rate is the number of low-to-high or high-to-low transitions made by the signal during a period of time.

The `-toggle_rate` option differs from the toggle rate used for modeling switching activity. The `-toggle_rate` option expresses the sum of the rise and fall transitions that the signal makes during an entire simulation, clock period, or other period you specify. Power Compiler uses the `-toggle_rate` and `-period` (or `-clock`) options to determine the actual toggle rate per unit of time.

The following example specifies that the net `net1` is at logic 1 for 20 percent of the time, and that it transitions between logic values 0 and 1 an average of 10 times in 1000 time units. The time unit used for the toggle rate is the time unit defined in the target library. The `-period` option is optional and defaults to a value of 1, when it is not specified.

```
dc_shell> set_switching_activity [get_nets net1] \
    -static_probability 0.2 -toggle_rate 10 -period 1000
```

Use the `-state_condition` option to annotate state-dependent toggle rates on pins or state-dependent static probabilities on cells. The state-dependent toggle rates can be annotated only if the library is characterized with state-dependent power tables for internal power, for the pins of the library cell. Similarly, state-dependent static probabilities can be annotated only if the library is characterized with state-dependent power tables for leakage power, for the library cells.

The following example shows how to use the `-state_condition` option to annotate the state-dependent toggle rates on pins. It specifies that the pin `ff1/Q` toggles 0.01 times when the pin `D` is at logic 1, and 0.03 times when the pin `D` is at logic 0.

```
dc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.01 \
    -state_condition "D"
dc_shell> set_switching_activity [get_pins ff1/Q] -toggle_rate 0.03 \
    -state_condition "!D"
```

Use the `-rise_ratio` option to specify the ratio of rise transitions to the total transitions for the specified toggle rate. You can also use this option with state-dependent toggle rates to specify the ratio of rise transitions to fall transitions for the specified state. The following example specifies that the `xor1/Y` pin toggles 0.01 times when the cell is in `A` state, and that 90 percent of these toggles are rise toggles.

```
dc_shell> set_switching_activity [get_pins xor1/Y] -toggle_rate 0.01 \
    -state_condition "A" -rise_ratio 0.9
```

Use the `-path_sources` option to specify the path-dependent toggle rates. The following example specifies that the `and1/Y` pin toggles 0.02 times because of a toggle on the input pin `A`, but never toggles because of a toggle on the `B` pin. Toggle rates that are both state-

and path-dependent are specified by using the `-state_condition` and `-path_sources` options together.

```
dc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.02 \
    -path_sources "A"
dc_shell> set_switching_activity [get_pins and1/Y] -toggle_rate 0.00 \
    -path_sources "B"
```

The state-dependent static probabilities can be annotated using the `-state_condition` option. The following example specifies that the cell named AND1 is at the A & B state for 10 percent of the time, at the A & !B state for 70 percent of the time, and at the !A state for 20 percent of the time.

```
dc_shell> set_switching_activity [get_cells AND1] \
    -static_probability 0.1 -state_condition "A & B"
dc_shell> set_switching_activity [get_cells AND1] \
    -static_probability 0.7 -state_condition "A & !B"
dc_shell> set_switching_activity [get_cells AND1] \
    -static_probability 0.2 -state_condition "!A"
```

To implicitly select outputs or cells to annotate, use the `-type` option and specify a list of following types of objects:

- Input, output, inout ports of design or input, output, inout pin of hierarchical cells
- Output of registers, output of sequential cells
- Output of black box cells
- Output of tristate cells
- Output of flip-flops clocked by the specified clocks
- Output of clock-gating cells
- Output of memory cells
- Nets

When you use the `set_switching_activity` command to annotate switching activity on all inputs, this includes the clock inputs as well. This results in overriding the switching activity on the clock inputs. To avoid overriding the switching activity on clock inputs, specify all inputs except the clock inputs, as shown in the following example:

```
dc_shell> set_switching_activity [remove_from_collection \
    [all_inputs] clk] \
    -static_probability sp_value -toggle_rate tr_value \
    -period period_value
```

For more information, see the `set_switching_activity` command man page.

Fully Versus Partially Annotating the Design

For the highest accuracy of power analysis, annotate all the elements in your design. To annotate all design elements, you must use gate-level simulation to monitor all the nodes of the design.

Using gate-level simulation, you can perform the following activities:

- Capture state- and path-dependent switching activity
- Capture switching activity that considers glitching (full-timing gate-level simulation only)

After layout, you can increase accuracy further by annotating wire loads with more accurate net capacitance values. However, if the design layout is performed at the foundry, you might not have access to the post-layout information.

If you annotate some design elements, Power Compiler uses an internal zero-delay simulation to propagate switching activity through nonannotated nets in your design. Power Compiler uses internal simulation anytime it encounters nonannotated nets during power analysis.

During switching activity propagation, Power Compiler tracks which design elements are user-annotated with the `set_switching_activity` command and which are not. In calculating power, Power Compiler does not overwrite user-annotated switching activity with propagated switching activity.

Power analysis and optimization require that you annotate at least the following:

- Primary inputs
- Outputs of synthesis-invariant elements such as black box cells
- Three-state devices
- Sequential elements
- Hierarchical ports

Note:

When performing power analysis on a partially annotated design, it is recommended that you specify a clock before running the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute and propagate switching activity through the design. Use the `create_clock` command to create a clock. If no clock is available, you get a PWR-80 warning message. This does not stop propagation but the estimated switching activity might not be accurate.

Analyzing the Switching Activity Annotation

The `report_saif` command can be used to display information about the annotated switching activity. The report generated by this command shows the number and percentage of nets, ports, and pins annotated with user-annotated switching activity, default switching activity, and propagated switching activity, respectively. The command does not consider clock-gating cells as synthesis-invariant because these cells can be deleted or inserted during the optimization step. The following example shows the report generated by the command:

```
dc_shell> report_saif -hierarchy
```

```
*****
Report : saif
        -hier
Design : des
Version: ...
Date   : ...
*****
```

Object type	User Annotated (%)	Default Annotated (%)	Propagated Activity (%)	Total
Nets	251(99.21%)	1(0.40%)	1(0.40%)	253
Ports	59(98.33%)	1(1.67%)	0(0.00%)	60
Pins	251(99.60%)	0(0.00%)	1(0.40%)	252

If the `-hier` option is used, the switching activity information is generated for all design objects in the design hierarchy starting from the current instance. If this option is missing, then only design objects in the hierarchical level of the current instance are considered.

If the `-rtl_saif` option is used, switching activity information for RTL-invariant objects is reported. Otherwise switching activity information for all design nets, ports, and pins are reported. You can use the `-rtl_saif` option after reading an RTL SAIF file.

The `-missing` option can be used to display the design objects that do not have user-annotated switching activity information.

Removing the Switching Activity Annotation

Switching activity annotation can be removed from all current design objects using the `reset_switching_activity` command. This command removes all the simple and state- and path-dependent switching activity information.

In the following example, an RTL SAIF file is read before a design is compiled with power constraints and then a more accurate gate-level SAIF file is used to generate power reports:

```
read_saif -map_names -input rtl.back.saif -instance_name tb_rtl/i
compile_ultra
...
reset_switching_activity
read_saif -input gate.back.saif -instance_name tb_gate/i
report_power
```

Note that in this example, the SAIF map is already initialized.

You can selectively remove the switching activity information from individual design objects using the following command:

```
dc_shell> set_switching_activity objects
```

Design Objects Without Annotated Switching Activity

Power Compiler needs switching activity information for all design nets and state- and path-dependent information for all design cells and pins to calculate power. Switching activity that is not user annotated is estimated automatically before power is calculated. This is performed in the following stages:

- The user-annotated and default-annotated switching activities are used to derive the simple static probability and toggle rate information for the rest of the design nets.
- The simple switching activity information (user-annotated or estimated) is used to derive the non-annotated state- and path-dependent switching activity.

Default Switching Activity Values

The following types of nets are automatically annotated with switching activity based on the logic of the design:

- Nets driven by constants: A toggle rate value of 0.0 is used. A static probability value of 0.0 is used for logic 0 constants, while a value of 1.0 is used for logic 1 constants.
- Nets driven by clocks: The toggle rate and static probability are derived from the clock waveform.
- Nets driving or driven by buffers: The switching activity of a nonannotated buffer input or output is set to match the switching activity already determined for the other side of the buffer.

- Nets driving or driven by inverters: The switching activity of the inverter input or output is based on the switching activity already determined for the other side of the inverter. The toggle rate is the same and the static probability is complementary.
- Flip-flop outputs: If a flip-flop cell has both Q and QN output ports and only one of the outputs is annotated, then the other output is assigned the same toggle rate and the complementary static probability.
- Inputs and outputs of black box cells: The switching activity cannot be propagated through a black box. Therefore, the default switching activity is annotated on the outputs of a black box.

The default switching activity depends on the value of the `power_default_static_probability` and `power_default_toggle_rate` variables. The default static probability is 0.5. To specify a different value, set the `power_default_static_probability` variable to the desired value.

The default toggle rate is 0.1 multiplied by the related clock frequency specified by the `-clock` option of the `set_switching_activity` command. In other words, the net is assumed to toggle once every 10 clock periods on average. If no related clock is specified for a net, the clock with the highest frequency is used. To specify a different toggle rate multiplier, set the `power_default_toggle_rate` variable to the desired multiplier value (default 0.1).

Propagating the Switching Activity

For nets that are not user-annotated and not assigned switching activity information by default, the tool uses a zero-delay simulator to propagate switching activity from known nets. Random simulation vectors are generated for the user and default annotated nets depending on the annotated toggle rate and static probability values. The zero-delay simulator uses the functionality of the design cells and the random vectors to obtain the switching activity on nonannotated cell outputs.

The number of simulation steps performed by this mechanism depends on the analysis effort option applied to the `report_power` command. User and default annotated switching activity values are never overwritten by values derived by the propagation mechanism.

However, if a design net is not annotated with both toggle rate and static probability values, then the switching activity on this net cannot be used by the propagation mechanism. For such nets, the nonannotated value is estimated by the propagation mechanism.

Deriving the State- and Path-Dependent Switching Activity

If an RTL SAIF file or a gate-level SAIF file without state- and path-dependent switching information is used to annotate the design switching activity, Power Compiler needs to estimate the required state- and path-dependent switching activity information. After obtaining the simple switching activity (from user annotation, or by switching activity propagation), Power Compiler estimates the state-dependent static probability information for every cell, and the state- and path-dependent toggle rate information for every cell pin. This information is obtained from the switching activities of each cell input and output pins. Although the state- and path-dependent estimation mechanism produces accurate power calculations, for best power results, use the gate-level SAIF files with state- and path-dependent information.

6

Performing Power Analysis

The information in this chapter describes the Power Compiler power analysis engine and how to perform power analysis.

This chapter contains the following sections:

- [Overview](#)
- [Identifying Power and Accuracy](#)
- [Performing Gate-Level Power Analysis](#)
- [Analyzing Power With Partially Annotated Designs](#)
- [Power Correlation](#)
- [Analyzing the Design For Power Analysis](#)
- [Characterizing a Design for Power](#)
- [Reporting the Power Attributes of Library Cells](#)
- [Power Reports](#)

Overview

After capturing switching activity, mapping your design to gates, and annotating your design, run the `report_power` command to report the power consumption of the various elements of the design.

The tool creates power reports for

- Design
- Modules
- Nets
- Cells or groups of cells of specific type
- Scenarios, in case of multicorner-multimode designs

The `report_power` command uses a Power Compiler license. When the command is completed, the license is released. If a license is not available, the command terminates with an error message.

To keep the license after completing the `report_power` command, set the following variable:

```
dc_shell> set_app_var power_keep_license_after_power_commands true
```

For more information see the command man page.

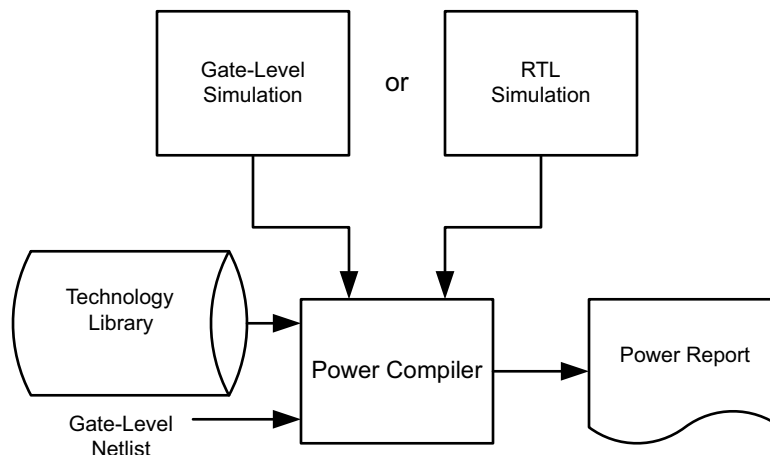
Identifying Power and Accuracy

Power Compiler uses different methods to compute the power of your design. The tool considers the type and amount of switching activity annotated on your design and chooses the most accurate method to compute your design's power. The method used depends on whether you annotate some or all of the elements in your design.

To analyze your gate-level design, the following inputs are required:

- Switching activity
- Logic library
- Gate-level netlist

[Figure 6-1](#) shows the inputs to Power Compiler.

Figure 6-1 Inputs to Power Compiler

For best results, use the logic libraries characterized with power information. If the library has only pin capacitance and voltage, but no power information, only the switching power of the net is reported. The switching power is a function of the pin capacitance, voltage, and toggle frequency. You can generate the report by using the `report_power` command. The power number reported corresponds to the switching power of the net, which is a function of the pin capacitance, voltage, and toggle frequency.

Factors Affecting the Accuracy of Power Analysis

The following factors can affect the accuracy of power analysis:

- Switching activity annotation
- Delay model
- Correlation
- Clock tree buffers
- Complex cells

Switching Activity Annotation

Annotating switching activity relies on the ability to map the names of the synthesis invariant objects in the RTL source to the equivalent object names in the gate-level netlist. Mapping inconsistencies can cause the SAIF file to be incorrectly or incompletely annotated, which

can affect the power analysis results. In turn, the quality of these results affects the results of power optimizations that rely on the annotation. For more information, see [Annotating Switching Activity Using RTL SAIF Files](#).

Clock Frequency Scaling

If a design is synthesized at a frequency that is different from the frequency the simulation is run, the SAIF file generated from the simulation reflects this difference. This causes a mismatch in timing and affects dynamic power analysis.

To correct this problem, the Power Compiler tool allows you to scale the clock frequency, resulting in a more accurate dynamic power analysis.

To enable clock frequency scaling, set the `power_enable_clock_scaling` variable to `true`. Then, run the `set_power_clock_scaling` command to specify clock scaling for power analysis. The command accepts either a `-period` option which specifies the clock period to which the clock is to be scaled, or a `-ratio` option which specifies a scaling value to be applied to the toggle rates of pins and nets.

When you use the `set_power_clock_scaling` command, the tool scales only the switching activity applied with the `read_saif` command. The tool does not scale the following switching activity applied with the `set_switching_activity` command.

For more details about this command, see the `set_power_clock_scaling` man page.

Delay Model

Power Compiler uses a zero-delay model for internal simulation and for propagation of switching activity during power analysis. This zero-delay model assumes that the signal propagates instantly through a gate with no elapsed time.

The zero-delay model has the advantage of enabling fast and relatively accurate estimation of power dissipation. The zero-delay model does not include the power dissipated due to glitching. If your power analysis must consider glitching, use power analysis after annotating switching activity from full-timing gate-level simulation. As mentioned previously, the internal simulation is used only for nodes that do not have user-annotated switching activity.

Switching Activity Propagation and Accuracy

While propagating switching activity through the design, the logic states of inputs of the gates' can have interdependencies that affect the accuracy of any statistical model. Such interdependency of inputs is called correlation. Correlation affects the accuracy of propagation of toggle rates. Because accurate analysis depends on accurate toggle rates, correlation also affects the accuracy of power analysis.

Power Compiler considers correlation within combinational and sequential logic, resulting in more accurate analysis of switching activity for many types of designs. The types of circuits

that exhibit high internal correlation are designs with reconvergent fanouts, multipliers, and parity trees. However, Power Compiler has no access to information about correlation external to the design. If correlation exists between the primary inputs of the design, Power Compiler does not recognize the correlation.

Power Compiler considers correlation only within certain memory and CPU thresholds, beyond which correlation is ignored. As the design size increases, Power Compiler reaches its memory limit and is not able to fully consider all internal correlation.

As an example of correlation, consider a 4-bit arithmetic logic unit (ALU) that performs five instructions. The data bus is 4-bits wide, and the instruction opcode lines are 3-bits wide. The assumption of uncorrelated inputs holds up well for the data bus lines inputs but fails for the opcode inputs if some instructions are used more often.

If your design has black boxes, such as complex cells, RAM, ROM, or macro cells you can annotate switching activity at the outputs of these elements.

Overriding Library Power Characterization

The `set_cell_internal_power` command sets or removes the `power_value` attribute on or from specified pins. The `power_value` attribute represents the power consumption for a single toggle of the pin. If a cell has at least one such annotated pin, its internal power is calculated by summing the annotated power values times the pin toggle rates. If the command is issued without setting the `power_value` attribute, the existing `power_value` attributes are removed from the specified pins. If the `power_value` attribute is specified without unit, the power unit of the library is used. If the library does not have a defined unit, an error message is issued.

Use the `set_cell_internal_power` command to override a cell's library power characterization in situations where that characterization does not apply; most commonly, when you manually replace a logic with a single cell and want the single cell's power consumption to represent the replaced logic. For example, if you replace a clock tree by a single buffer cell, you can set the `power_value` attribute on the output pin of the buffer cell with the value of the power consumption for one clock toggle of the entire clock tree. Although the library cell is characterized, its power consumption is calculated using the value of the `power_value` attribute set by the `set_cell_internal_power` command.

For more information, see the command man page.

Performing Gate-Level Power Analysis

After annotating your design with switching activity, use the `report_power` command to report the power of your gate-level design.

To perform power analysis on a partially annotated design, specify a clock before invoking the `report_power` command. The internal zero-delay simulation requires a real or virtual clock to properly compute switching activity. Use the `create_clock` command to create the clock.

Using the `report_power` Command

The `report_power` command calculates and reports power for a design.

When you do not annotate switching activity on the nets, the command performs zero-delay simulation to propagate switching activity for the nets. To compute the switching activity for internal nets, the command uses the switching activity for startpoint nets (if available). The nets that are annotated using the `set_switching_activity` or the `read_saif` command are not overwritten during the switching activity propagation.

If you annotate switching activity on all the elements of the design, Power Compiler does not propagate any switching activity through the design. Instead, power analysis uses the annotated gate-level switching activity.

Command options enable you to print with different sorting modes and with verbose and cumulative options. The default operation is to print a power summary for the instance's subdesign (in the context of the higher-level design).

Power analysis uses any net loads during the power calculation. For nets that do not have back-annotated capacitance, Power Compiler estimates the net load from the appropriate wire load model from the logic library. If you have annotated any cluster information about the design using Synopsys Floorplan Manager, Power Compiler uses the improved capacitance estimates from the cluster's wire loads.

In the topographical mode the `report_power` command reports the correlated power of the design as a sum of estimated clock tree power and netlist power. For more details see [Power Reports](#).

The `report_power` command has the following syntax:

```
report_power
  [-net]
  [-cell]
  [-groups list_of_cell_group]
  [-only cell_or_net_list]
  [-cumulative]
  [-flat]
  [-exclude_boundary_nets]
  [-include_input_nets]
  [-analysis_effort low | medium | high]
  [-verbose]
  [-nworst number]
  [-sort_mode mode]
  [-histogram]
```

```

[-exclude_leq le_val]
[-exclude_geq ge_val]
[-nosplit]
[-hierarchy]
[-levels level_value]
[-scenarios {scenario_name1 scenario_name_2 ...}]

```

The `report_power` command calculates and reports static and dynamic power for the current design. It uses the user-annotated switching activity to calculate the net switching power, cell internal power, and cell leakage power. When you do not specify any option, by default, the `report_power` command displays the summary of power values only for the current design. If you specify a cell instance, the command reports the summary power values for the specified instance. The command supports several options, for you to specify cells, nets, scenarios, include or exclude boundary nets, and so on. The list of options are:

`-sort_mode mode`

The `report_power -cell` command uses `cell_internal_power` as the default for the `-sort_mode` option. If the logic library does not have any internal power modeling for leaf cells, `report_power -cell -nworst 10`, for example, retrieves only the first ten cells (alphabetically).

When you use the `report_power -net` command, `net_switching_power` is the default for the `-sort_mode` option. If both the `-net` and `-cell` options are specified and a sort mode is explicitly specified, the selected sort mode is used for both the cell and net reports. Therefore, the selected sort mode is one of the sort modes that applies to both options. If both the `-net` and `-cell` options are specified, by default, the sort mode for `report_power` is total dynamic power.

`-histogram`

This option prints a histogram-style report with the number of nets in each power range. Use the `-exclude_leq` and `-exclude_geq` options, respectively, to exclude data values less than *le_val* or greater than *ge_val*. This option is useful for printing the range and variation of power in the design, and prints a histogram report only when used with the `-net` or `-cell` options.

`-groups list_of_cell_group`

This option reports power for the specified power groups. When you do not specify this option, by default, reports the predefined groups listed in [Table 6-1](#). The `-groups` option is mutually exclusive with the `-net`, `-hierarchy`, `-levels`, `-only`, and `-cumulative` options.

[Table 6-1](#) lists the power groups and the cell types that belong to the group, in the descending order of priority.

Table 6-1 Groups and Their Cell Types in the Descending Order of Priority

Group	Cell types belonging to the group
io_pad	Cells defined in the pad_cell group in the library
memory	Cells defined in the memory group in the library
black_box	Cells that do not have any functional description in the library
clock_network	Cells in the clock network, excluding the io_pad cells
register	Latches and flip-flops driven by the clock network, excluding the io_pad and black_box cells
sequential	Latches and flip-flops clocked by signals that are not in the clock network
combinational	Cells that have a functional description and are not sequential cells

-nosplit

Most of the design information is listed in fixed-width columns. If the information for a column exceeds its column width, the next column begins on a new line, starting in the correct column. This option prevents line splitting and facilitates scripts to extract information from the report output.

-hierarchy

This option enables you to view internal, switching, and leakage power consumed in your design hierarchy, on a block-by-block basis. The hierarchical levels of the design are indicated by indentations.

-levels *level_value*

Use this option only with the **-hierarchy** option. This option enables you to limit the depth of the hierarchy tree displayed in the report. The *level_value* setting should be an integer number greater than or equal to 1. For example, to see the power results for all blocks up to 2 levels from the top, use the following command:

```
dc_shell> report_power -hierarchy -levels 2
```


-scenarios

This option reports the power details for the specified list of scenarios for a multimode design. Inactive scenarios are not reported. When this option is not used, only the current scenario is reported.

For more information, see the `report_power` command man page.

Using the `report_power_calculation` Command

Power Compiler uses a complex mechanism to calculate dynamic and leakage power. The dynamic power consists of internal power on pins and switching power on nets. Both internal and leakage power could be state dependent.

Though the `report_power` command does provide a comprehensive report, it is often a mystery how the numbers relate to the power tables in the library.

The `report_power_calculation` command shows how the reported power numbers are derived from the various inputs such as library, simulation data, netlist, and parasitics. This command does not work on the libraries that have built-in security to protect the power table numbers. This restriction does not apply for switching power.

For more information, see the `report_power_calculation` command man page.

Analyzing Power With Partially Annotated Designs

If you invoke power analysis without annotating any switching activity, Power Compiler uses the following defaults for the primary inputs of your design:

- $P_1 = 0.5$ (the signal is in the 1 state 50 percent of the time)

P_1 is the probability that input P is at logic state 1. For definitions of static probability, P_1 , and toggle rate (TR), see [Types of Switching Activity to Annotate](#).

- $TR = 0.1 * f_{clk}$ (the signal switches one time, every 10 clock cycles)

f_{clk} is the frequency of the input's related clock in the design, as defined by the `set_switching_activity` command. You can specify the related clock explicitly with its clock name or implicitly as “*”. In the latter case, Power Compiler infers a related clock automatically. If the input port does not have a related clock, Power Compiler uses the fastest clock in the design.

Using the defaults for static probability and toggle rate can be reasonable for data bus lines. However, the defaults might be unacceptable for some signals, such as a reset or a test-enable signal.

If you neglect to annotate toggle information about primary inputs, these inputs assume the default toggle value. If the input or logic connected to this input is heavily loaded, the results could be significantly different from what you expect.

To change the default for switching activity and static probability, set the following variables:

- `power_default_static_probability`
This variable sets the default for static probability.
- `power_default_toggle_rate`
This variable sets the default for toggle rate.
- `power_default_toggle_rate_type`
The default is `fastest_clock`, which causes Power Compiler to calculate the default toggle rate by multiplying the fastest clock frequency with the default toggle rate. Set this variable to `absolute` to determine the behavior when the design object does not have a specified related clock; the tool uses the value of the `power_default_toggle_rate` variable.

The variables remain in effect throughout the `dc_shell` session in which you set them.

The following example sets the default static probability to 0.3:

```
dc_shell> set_app_var power_default_static_probability 0.3
```

The following example sets the default toggle rate to 0.4 of the toggle rate of the highest-frequency clock:

```
dc_shell> set_app_var power_default_toggle_rate 0.4
```

Power Correlation

Power correlation refers to the relationship between two power calculations: power after logic synthesis and power after place and route. Power after place and route is the final power, and you might want to know this number early in the process so you can take corrective action if the number exceeds your limits. Power correlation is supported only in Design Compiler topographical mode.

In `dc_shell`, the power reported after logic synthesis is often significantly different from the final power, and is, therefore, not a good predictor for final power. This differential is caused by three factors:

- Logic synthesis uses wire load models.
- High fanout nets are not synthesized.
- Clock trees do not exist in the design at the time of synthesis.

Performing logic synthesis within the Design Compiler topographical domain shell addresses the first two factors because this shell uses a virtual layout, not wire load models, and high fanout nets are synthesized automatically.

You specify to perform clock-tree estimation within `dc_shell-topo` to eliminate the differential caused by the third factor.

To improve correlation in cases with abnormal floor plans, you should use the physical constraints extracted from the floor plan.

Performing Power Correlation

Correlated power refers to the design power that is added to the estimated clock-tree power after logic synthesis in the Design Compiler topographical mode. Correlated power is also referred as estimated total power.

To calculate the correlated power, enable the power prediction feature by using the `set_power_prediction` command.

The syntax of the `set_power_prediction` command is:

```
set_power_prediction true | false  
    [-ct_references list_of_buffers_and_inverters]
```

Specify the clock tree references by using the `-ct_references` option, to perform clock-tree estimation which improves the correlation results.

When the power prediction feature is enabled, the `report_power` command reports the correlated power after the design has been mapped to technology-specific cells. When the power prediction feature is disabled, the `report_power` command reports only the total power, static power, and dynamic power, without considering the estimated clock-tree power.

The power prediction setting is also saved with the design, when the design is saved in the `.ddc` (Synopsys logical database format) binary file format.

Power Correlation Script

The following example script correlates power after you have setup your design environment and applied synthesis constraints:

```
read_verilog  
set_power_prediction  
compile_ultra  
report_power  
write -format ddc -output design.ddc
```

In Design Compiler topographical mode, the `report_power` command reports estimated total power, which includes the clock-tree contributions for internal, net-switching, and leakage power.

Analyzing the Design For Power Analysis

Follow these steps to get quick results from gate-level power analysis:

1. Create a SAIF file.

This step requires RTL simulation. For information, see [Generating SAIF Files](#).

2. Compile the design to gates, using various suitable options.
3. Annotate switching activity on primary inputs and other synthesis-invariant elements of the gate-level design.

For information about using SAIF files from RTL simulation to annotate switching activity, see [Generating SAIF Files](#).

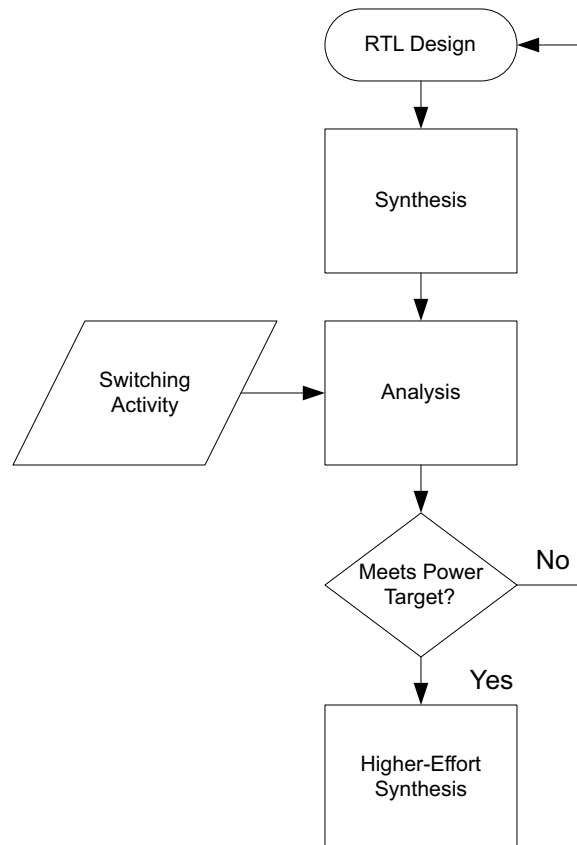
4. Use the `report_power` command to analyze your design's power.

Power Compiler uses an internal zero-delay simulation to propagate switching activity through nonannotated elements of the design.

5. Repeat steps 1 through 4 for other architectures and coding styles.

Quick gate-level power analysis enables you to see the results of changes in your RTL design.

[Figure 6-2](#) shows the steps that are followed in design exploration using Power Compiler.

Figure 6-2 Analyzing the Design for Power Analysis

After you refine your RTL design within the iterative loop of design exploration, your design is ready for a higher-effort synthesis.

Characterizing a Design for Power

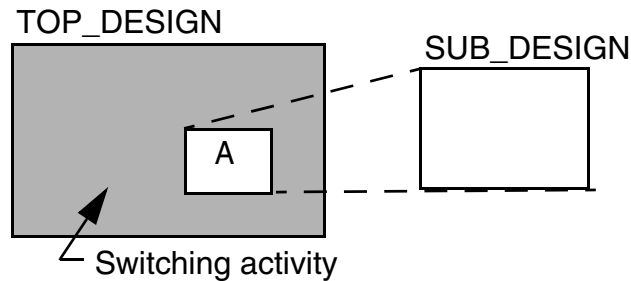
The `-power` option of the `characterize` command is useful in power analysis and optimization. This option characterizes annotated or propagated switching activity from the instance of a subdesign to the nets of the subdesign referenced by the instance. There must be a one-to-one correspondence between the nets in the instance and the nets in the referenced subdesign.

As shown in [Figure 6-3](#), consider a design hierarchy in which A is a design instance of SUB_DESIGN in TOP_DESIGN. Instance A references SUB_DESIGN. When you invoke

power analysis on TOP_DESIGN, the switching activity propagates throughout any nets that are not already user-annotated.

```
dc_shell> report_power top_design
```

Figure 6-3 Switching Activity for TOP_DESIGN

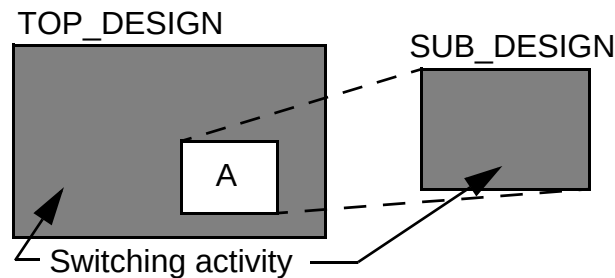


The switching activity can be propagated from primary inputs and synthesis-invariant elements. In this example, user-annotated on individual design elements using `set_switching_activity` commands, or both.

As shown in Figure 6-4, if you set the current instance to A and characterize for power, `characterize` writes the switching activity of instance A onto SUB_DESIGN.

```
dc_shell> current_design TOP_DESIGN
dc_shell> characterize A -power
```

Figure 6-4 Switching Activity for SUB_DESIGN



After characterizing, you can report the power of SUB_DESIGN by using the newly characterized switching activity. If you have Power Compiler, you can compile the SUB_DESIGN by using the newly characterized switching activity.

The `-power` option of `characterize` relies on a one-to-one correspondence between the nets of the referenced SUB_DESIGN and its instance A. If you compile the subdesign before characterizing instance A or make any changes that alter the nets or names of nets, the one-to-one net correspondence is lost and `characterize` fails.

After compiling a subdesign and before reanalyzing or compiling TOP_DESIGN, be sure to relink the designs.

Before recompiling the subdesign, follow some or all of the following steps:

- Relink the designs using link.
- Generate new switching activity for changed designs.
- Annotate or propagate new switching activity on designs.
- Characterize before reanalyzing or recompiling the subdesign.

For more information about the `characterize` command, see the Design Compiler documentation and the online man pages.

Reporting the Power Attributes of Library Cells

Use the `report_lib -power` command to report which library cells have power characterization and what type of characterization exists on each library cell. The `report_lib -power` command reports the following information for each cell:

- Leakage power attribute
- Internal power attribute
- Attribute for separate rise and fall power
- Attribute for average rise and fall power
- Toggling pin specified by the internal power table
- Any when conditions (for state-dependent power)
- The `related_pin` or `related_input` for path-dependent power

For more information about the command, see the `report_lib` command man page.

Power Reports

This section contains examples of reports generated with the `report_power` command and various combinations of report options.

The `report_power` command in topographical mode reports the correlated power, consisting of estimated clock tree power and netlist power. If the tool cannot perform clock

tree estimation, Power Compiler issues a warning that the clock tree estimation could not be performed.

Examples of power reports using the various options of the `report_power` command are described in the following sections:

- [Power Report Summary](#)
- [Net Power Report](#)
- [Cell Power Report](#)
- [Group Report](#)
- [Hierarchical Power Reports](#)
- [Power Report for Block Abstraction](#)

Power Report Summary

[Example 6-1](#) shows a power report summary.

Example 6-1 Summary Report of the report_power Command

```
dc_shell> report_power -analysis_effort high -verbose
*****
Report : power
        -analysis_effort high
        -verbose
Design : DESIGN_1
Version: A-2007.12-SP2
Date   : Fri Feb 22 01:46:34 2008
*****
Library(s) Used:
        slow (File: slow.db)

Operating Conditions:
Wire Loading Model Mode: Inactive

Global Operating Voltage = 1.62
Power-specific unit information :
    Voltage Unit = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW      (derived from V,C,T units)
    Leakage Power Units = 1nW

Cell Internal Power Breakdown
-----
Combinational      = 3.0975 mW   (10%)
Sequential         = 22.3222 mW  (72%)
Other              = 0.0000 mW   (0%)

Combinational Count = 13470
Sequential Count    = 2382
Other Count         = 0
Information: Reporting correlated power. (PWR-620)

Cell Internal Power = 27.2572 mW   (76%)
Net Switching Power = 8.6208 mW   (24%)
-----
Total Dynamic Power = 35.8779 mW (100%)
Cell Leakage Power  = 2.6586 uW

Power Breakdown
-----
```

Cell	Cell Internal Power (mW)	Driven Net Switching Power (mW)	Tot Dynamic Power (mW) (% Cell/Tot)	Cell Leakage Power (pW)
Netlist Power	25.4197	5.5186	3.094e+01 (82%)	2.649e+03
Estimated Clock Tree Power	1.8375	3.1021	4.9396 (37%)	9.9143

```
-----
```

Net Power Report

[Example 6-2](#) shows a net power report sorted by the net switching power and filtered to display only the five nets with the worst switching power.

Example 6-2 Net Power Report, Sorting, and Display Options

```
dc_shell> report_power -net -flat -sort_mode net_switching_power -nworst 5
```

```
*****
Report:  power
        -net
        -nworst 5
        -flat
        -sort_mode net_switching_power
Design:  DESIGN_1
Version: A-2007.12-SP2
Date    : Fri Feb 22 01:50:50 2008
*****
Library(s) Used:

        power_lib.db (File: /remote/libraries/power_lib.db)

Operating Conditions: slow   Library: slow
Wire Load Model Mode: Inactive.
```

```
Global Operating Voltage = 1.62
Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW      (derived from V,C,T units)
Leakage Power Units = 1nW
```

Attributes

- ```

a - Switching activity information annotated on net
d - Default switching activity information on net
```

| Net                                    | Total<br>Net Load | Static<br>Prob. | Toggle<br>Rate | Switching<br>Power | Attrs |
|----------------------------------------|-------------------|-----------------|----------------|--------------------|-------|
| U_TAP_DBG_U_DBG_net5051                | 0.463             | 0.374           | 0.1968         | 0.1195             |       |
| U_CORE/U_CONTROL_U_A7S_pencadd_net5225 | 0.248             | 0.374           | 0.1968         | 0.0641             |       |
| U_CORE/U_CONTROL_U_A7S_dataio_net5298  | 0.247             | 0.374           | 0.1968         | 0.0637             |       |
| U_CORE/U_MUL8_net5450                  | 0.232             | 0.374           | 0.1968         | 0.0599             |       |
| U_CORE/U_AREG_net5593                  | 0.194             | 0.374           | 0.1968         | 0.0501             |       |
| Total (5 nets)                         |                   |                 |                | 357.2614 uW        |       |

## Cell Power Report

[Example 6-3](#) displays a cell power report containing the cumulative cell power report. The cells are sorted by cumulative fanout power values, only the top five are reported.

### Example 6-3 Cell Power Report Containing Cumulative Cell Power

```
dc_shell> report_power -cell -analysis_effort low \
-sort_mode cell_internal_power

Report : power
 -cell
 -analysis_effort low
 -sort_mode cell_internal_power
Design : DESIGN_3
Version: B-2008.09
Date : Fri Aug 08 01:51:28 2008

Library(s) Used:

 slow (File: slow.db)

Operating Conditions: slow Library: slow
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62
Power-specific unit information :
 Voltage Units = 1V
 Capacitance Units = 1.000000pf
 Time Units = 1ns
 Dynamic Power Units = 1mW (derived from V,C,T units)
 Leakage Power Units = 1nW

Information: Reporting correlated power. (PWR-620)

Attributes

 h - Hierarchical cell
```

| Cell                                  | Cell<br>Internal<br>Power | Driven Net<br>Switching<br>Power | Tot Dynamic<br>Power<br>(% Cell/Tot) | Cell<br>Leakage<br>Power | Attrs |
|---------------------------------------|---------------------------|----------------------------------|--------------------------------------|--------------------------|-------|
| CLOCK_TREE_EST                        | 1.8375                    | 3.1021                           | 4.940 (37%)                          | 9.9144                   |       |
| U_CORE                                | 21.7118                   | N/A                              | N/A (N/A)                            | 2226.6487                | h     |
| U_TAP_DBG_U_DBG_clk_gate_int_en_d_reg | 0.0123                    | N/A                              | N/A (N/A)                            | 1.4392                   | h     |
| 0.0112 6.968e-04 1.19e-02 (94%)       |                           | 0.1458                           |                                      |                          |       |
| U_TAP_DBG_U_SCAN1_breakpt_in_d_reg    | 0.0106                    | 2.472e-04                        | 1.09e-02 (98%)                       | 0.1458                   |       |
| U_TAP_DBG_U_ID_REG_clk_gate_shift_reg |                           |                                  |                                      |                          |       |
| Totals (2474 cells)                   | 27.368mW                  | N/A                              | N/A (N/A)                            | 2.658uW                  |       |

## Group Report

[Example 6-4](#) shows the report generated by the `report_power` command when you use the `-groups` option.

### Example 6-4 Cell Report for Various Groups

```
dc_shell> report_power -groups "io_pad memory combinational"
```

\*\*\*\*\*  
Report : power

-analysis\_effort low  
Design : RISC\_CORE  
Version: F-2011.09-BETA5  
Date : Tue Aug 2 23:20:36 2011  
\*\*\*\*\*

Library(s) Used:  
tcbn65lpwc\_ccs (File: tcbn65lpwc\_pg.db)  
tcbn65lpwc0d720d9\_ccs (File: tcbn65lpwc0d720d9\_pg.db)  
tcbn65lpwc0d72\_ccs (File: tcbn65lpwc0d72\_pg.db)  
tcbn65lpwc0d90d72\_ccs (File: tcbn65lpwc0d90d72\_pg.db)

Global Operating Voltage = 1.08  
Power-specific unit information :  
Voltage Units = 1V  
Capacitance Units = 1.000000pf  
Time Units = 1ns  
Dynamic Power Units = 1mW (derived from V,C,T units)  
Leakage Power Units = 1nW

| Power Group   | Internal Power | Switching Power | Leakage Power | Total Power | ( % )     | Attrs |
|---------------|----------------|-----------------|---------------|-------------|-----------|-------|
| io_pad        | 0.0000         | 0.0000          | 0.0000        | 0.0000      | (0.00% )  |       |
| memory        | 0.0000         | 0.0000          | 0.0000        | 0.0000      | (0.00% )  |       |
| combinational | 6.0868e-02     | 7.1321e-02      | 777.9665      | 0.1330      | (100.00%) |       |
| Total         | 6.0868e-02 mW  | 7.1321e-02 mW   | 777.9665 nW   | 0.1330 mW   |           |       |

Net Switching Power = 71.3213 uW (53.95%)  
Cell Internal Power = 60.8676 uW (46.05%)  
-----

Total Dynamic Power = 132.1889 uW ( 100%)  
Cell Leakage Power = 777.9665 nW

1

## Hierarchical Power Reports

[Example 6-5](#) shows the results of the `report_power` command using the `-hierarchy` option. This option shows the internal, switching, and leakage power consumed in your design hierarchy, on a block-by-block basis.

### Example 6-5 Hierarchical Power Report

```
dc_shell> report_power -hierarchy
```

```

Report : power
 -hierarchy
 -analysis_effort low
Design : DESIGN_4
Version: A-2007.12-SP2
Date : Fri Feb 22 01:51:42 2008

```

Library(s) Used:

slow (File: slow.db)

Operating Conditions: slow Library: slow  
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62  
Power-specific unit information :  
Voltage Units = 1V  
Capacitance Units = 1.000000pf  
Time Units = 1ns  
Dynamic Power Units = 1mW (derived from V,C,T units)  
Leakage Power Units = 1nW

Information: Reporting correlated power. (PWR-620)

| Hierarchy         | Switch<br>Power | Int<br>Power | Leak<br>Power | Total<br>Power | %     |
|-------------------|-----------------|--------------|---------------|----------------|-------|
| A7S_top           | 8.683           | 27.368       | 2.66e+03      | 36.054         | 100.0 |
| CLOCK_TREE_EST    | 3.102           | 1.837        | 9.914         | 4.940          | 13.7  |
| U_CORE (A7S_core) | 4.318           | 21.712       | 2.23e+03      | 26.032         | 72.2  |

[Example 6-6](#) shows the results of the `report_power` command using the `-hierarchy` and `-levels` options. The `-hierarchy` option shows the internal, switching, and leakage power

consumed in your design hierarchy, on a block-by-block basis. The `-levels` option limits the depth of the hierarchy level displayed in the report.

*Example 6-6 Hierarchical Power Report With Specified Level of Hierarchy*

```
dc_shell> report_power -hierarchy -levels 1
```

```

Report : power
 -hierarchy
 -analysis_effort low
 -levels 2
Design : A7S_top
Version: A-2007.12-SP2
Date : Fri Feb 22 01:51:42 2008

```

Library(s) Used:

slow (File: slow.db)

Operating Conditions: slow Library: slow  
Wire Load Model Mode: Inactive.

Global Operating Voltage = 1.62

Power-specific unit information :

Voltage Units = 1V

Capacitance Units = 1.000000pf

Time Units = 1ns

Dynamic Power Units = 1mW (derived from V,C,T units)

Leakage Power Units = 1nW

Information: Reporting correlated power. (PWR-620)

| Hierarchy         | Switch<br>Power | Int<br>Power | Leak<br>Power | Total<br>Power | %     |
|-------------------|-----------------|--------------|---------------|----------------|-------|
| A7S_top           | 8.683           | 27.368       | 2.66e+03      | 36.054         | 100.0 |
| CLOCK_TREE_EST    | 3.102           | 1.837        | 9.914         | 4.940          | 13.7  |
| U_CORE (A7S_core) | 4.318           | 21.712       | 2.23e+03      | 26.032         | 72.2  |

## Power Report for Block Abstraction

When you use the `create_block_abstraction` command, the power information is saved as attributes on the block abstractions. If you annotate the switching activity, either by using the SAIF file or the `set_switching_activity` command, the switching activity is used to calculate the power information of the block abstractions, as shown in the following example:

```
Current design is 'test11_0'
create_block_abstraction
 internal power = 62.508907
 leakage power = 227754.921875
 net switching power = 17.909983, dyn_unit = 1mW, leak_unit = 1nW
```

By default, the `report_power` command reports the total power of the block, using the power information saved as attributes on the model, as shown in the following example:

```
...
Information: u_test11_3 test11_3 is block, internal power = 61.429829 mW
Information: u_test11_2 test11_2 is block, internal power = 60.608749 mW
Information: u_test11_1 test11_1 is block, internal power = 63.291779 mW
Information: u_test11_0 test11_0 is block, internal power = 62.508907 mW
Information: u_test11_core is block, leakage power = 122.281441 uW
Information: u_test11_3 test11_3 is block, leakage power = 222.416214 uW
Information: u_test11_2 test11_2 is block, leakage power = 224.137466 uW
Information: u_test11_1 test11_1 is block, leakage power = 228.666733 uW
Information: u_test11_0 test11_0 is block, leakage power = 227.754929 uW
Information: u_test11_3 test11_3 is block, net switching power =
17.291439mW
Information: u_test11_2 test11_2 is block, net switching power =
17.869442 mW
Information: u_test11_1 test11_1 is block, net switching power =
18.588411 mW
Information: u_test11_0 test11_0 is block, net switching power =
17.909983 mW
```

| Power Group   | Internal Power | Switching Power | Leakage Power | Total Power   | ( % ) Attrs |
|---------------|----------------|-----------------|---------------|---------------|-------------|
| io_pad        | 0.0000         | 0.0000          | 0.0000        | 0.0000        | (0.00%)     |
| memory        | 0.0000         | 0.0000          | 0.0000        | 0.0000        | (0.00%)     |
| black_box     | 271.0595       | 85.7838         | 1.0253e+06    | 357.8686      | (18.53%)    |
| clock_network | 0.9882         | 1.5616e+03      | 117.9128      | 1.5625e+03    | (80.89%)    |
| register      | 3.9093         | 0.3665          | 2.3101e+04    | 4.2989        | (0.22%)     |
| sequential    | 0.0000         | 0.0000          | 0.0000        | 0.0000        | (0.00%)     |
| combinational | 1.2514         | 5.6117          | 5.5759e+04    | 6.9189        | (0.36%)     |
| Total         | 277.2083 mW    | 1.6533e+03 mW   | 1.1042e+06 nW | 1.9316e+03 mW |             |

1

For more information about block abstractions, see the *Design Compiler User Guide*.





## Part 3: Power Reduction

---



# 7

## Clock Gating

---

Power optimization at higher levels of abstraction has a significant impact on reduction of power in the final gate-level design. Clock gating is an important technique for reducing the power consumption of a design.

This chapter includes the following sections:

- [Introduction to Clock Gating](#)
- [Using Clock-Gating Conditions](#)
- [Inserting Clock Gates](#)
- [Clock Gating Flows](#)
- [Ensuring Accuracy When Using Ideal Clocks](#)
- [Specifying Clock-Gate Latency](#)
- [Calculating the Clock Tree Delay From Clock-Gating Cell to Registers](#)
- [Specifying Setup and Hold](#)
- [Clock-Gating Styles](#)
- [Modifying the Clock-Gating Structure](#)
- [Integrated Clock-Gating Cells](#)
- [Clock-Gating Naming Conventions](#)

- [Keeping Clock-Gating Information in a Structural Netlist](#)
- [Replacing Clock-Gating Cells](#)
- [Clock-Gate Optimization Performed During Compilation](#)
- [Performing Clock-Gating on DesignWare Components](#)
- [Reporting Command for Clock Gates](#)

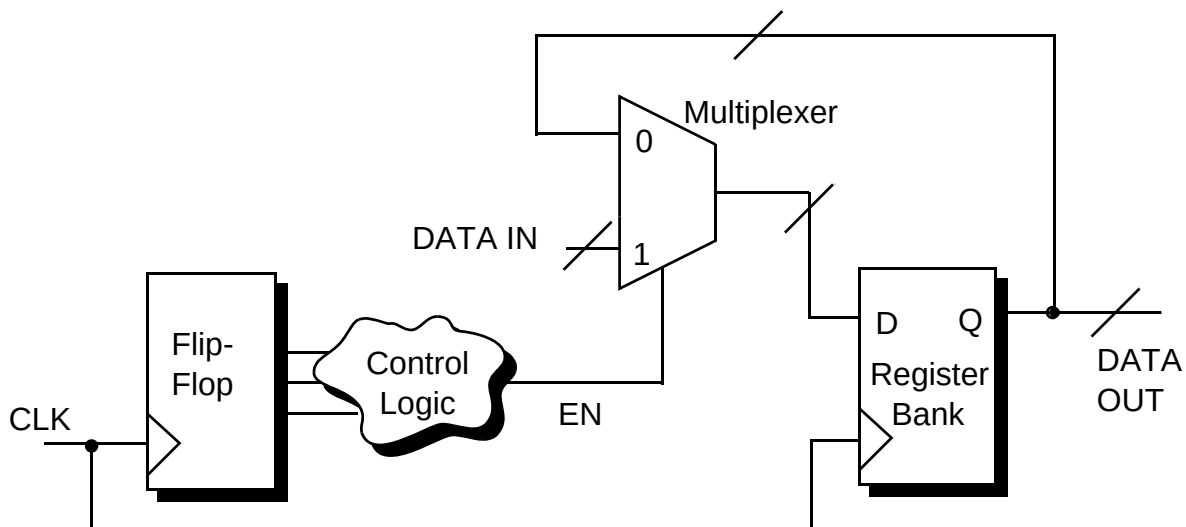
## Introduction to Clock Gating

Clock gating applies to synchronous load-enable registers, which are flip-flops that share the same clock and synchronous control signals. Synchronous control signals include synchronous load-enable, synchronous set, synchronous reset, and synchronous toggle.

Synchronous load-enable registers are represented by a register with feedback loop which maintains the same logic value through multiple cycles. Clock gating applied to synchronous load enable registers reduces the power needed when reloading the register banks.

[Figure 7-1](#) shows a simple register bank implementation using a multiplexer with feedback loop.

*Figure 7-1 Synchronous Load-Enable Register With Multiplexer*



When the synchronous load enable signal (EN) is at logic state 0, the register bank is disabled. In this state, the circuit uses the multiplexer to feed the Q output of each storage element in the register bank back to the D input. When the EN signal is at logic state 1, the register is enabled, allowing new values to load at the D input.

Such feedback loops can unnecessarily use power. For example, if the same value is reloaded in the register throughout multiple clock cycles (EN equals 0), the register bank and its clock net consume power while values in the register bank do not change. The multiplexer also consumes power.

Clock gating eliminates the feedback net and multiplexer shown in [Figure 7-1](#) by inserting a gate in the clock net of the register.

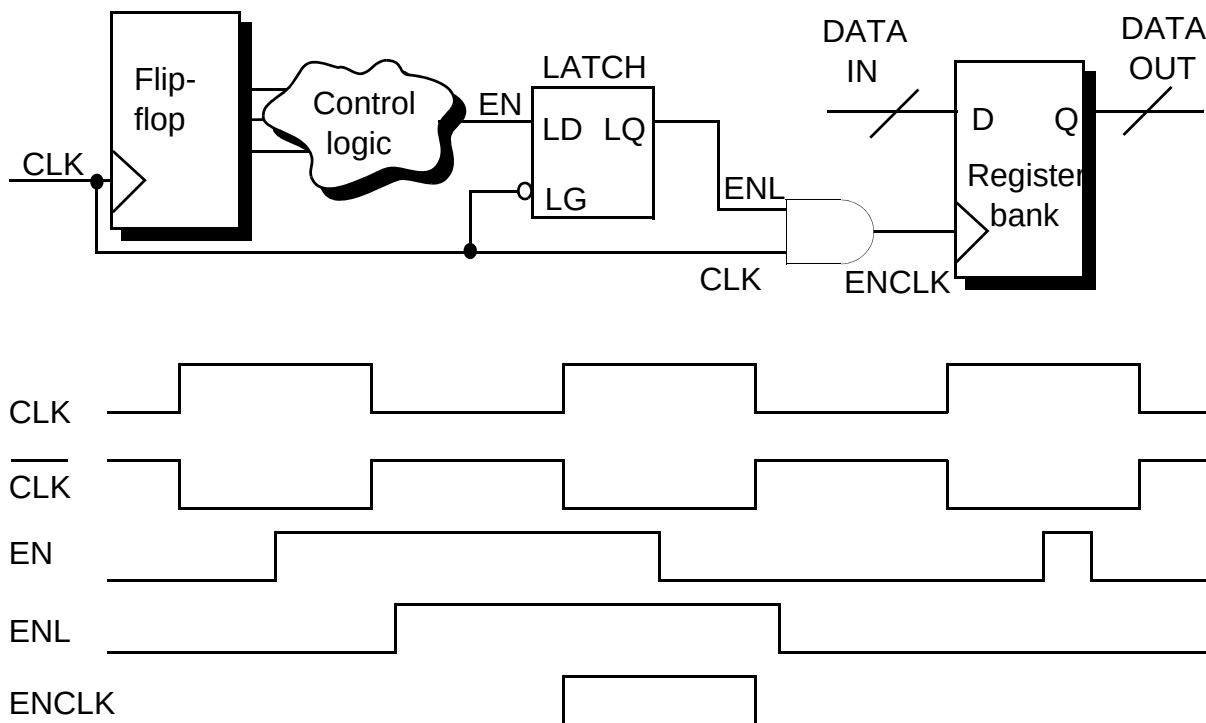
**Note:**

While applying the clock-gating techniques, the tool considers generated clocks similar to defined clocks.

The clock-gating cell selectively prevents clock edges, thus preventing the gated-clock signal from clocking the gated register.

[Figure 7-2](#) shows a latch-based clock-gating cell and the waveforms of the signals are shown with respect to the clock signal, CLK.

*Figure 7-2 Latch-Based Clock Gating*



The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it derives from the EN signal on the multiplexer shown in [Figure 7-1 on page 7-3](#). The register bank is triggered by the rising edge of the ENCLK signal.

The latch prevents glitches on the EN signal from propagating to the register's clock pin. When the CLK input of the 2-input AND gate is at logic state 1, any glitching of the EN signal could, without the latch, propagate and corrupt the register clock signal. The latch eliminates this possibility because it blocks signal changes when the clock is at logic state 1.

In latch-based clock gating, the AND gate blocks unnecessary clock pulses by maintaining the clock signal's value after the trailing edge. For example, for flip-flops inferred by HDL

constructs of rising-edge clocks, the clock gate forces the gated clock to 0 after the falling edge of the clock.

By controlling the clock signal for the register bank, you can eliminate the need for reloading the same value in the register through multiple clock cycles. Clock gating inserts clock-gating circuitry into the register bank's clock network, creating the control to eliminate unnecessary register activity.

Clock gating reduces clock network power dissipation, relaxes datapath timing, and reduces routing congestion by eliminating feedback multiplexer loops. For designs that have large register banks, clock gating can save power and area by reducing the number of gates in the design. However, for smaller register banks, the overhead of adding logic to the clock tree might not compare favorably to the power saved by eliminating a few feedback nets and multiplexers.

---

## Using Clock-Gating Conditions

Before gating the clock signal of a register, Power Compiler checks to see if certain clock-gating conditions are satisfied. Power Compiler inserts a clock gate only if all the clock-gating conditions are satisfied:

- The circuit demonstrates synchronous load-enable functionality.
- The circuit satisfies the setup condition.
- The register bank or group of register banks satisfies the minimum number of bits you specify with the `set_clock_gating_style -minimum_bitwidth` command. The default minimum bitwidth is 3.

After clock gating is complete, the status of clock-gating conditions for gated and ungated register banks appears in the clock-gating report. For information about the clock-gating report, see [“Reporting Command for Clock Gates” on page 7-78](#).

---

## Clock-Gating Conditions

The register must satisfy the following conditions for Power Compiler to gate the clock signal of the registers:

- Enable condition

If the register bank's synchronous load-enable signal is a constant logic 1, reducible to logic 1, or logic 0, the condition is `false` and the circuit is not gated. If the synchronous load-enable signal is not a constant logic 1 or 0, the condition is `true` and the setup condition is checked. The enable condition is the first condition that the tool checks.

- Setup condition

This condition applies to latch-free clock gating only. The enable signal must come from a register that uses the same clock as the register being gated. The setup condition is checked only if the register satisfies the enable condition.

- Width condition

The width condition is the minimum number of bits for gating registers or groups of registers with equivalent enable signals. The default is 3. You can set the width condition by using the `-minimum_bitwidth` option of the `set_clock_gating_style` command. The width condition is checked only if the register satisfies the enable condition and the setup condition.

## Enable Condition

The enable condition of a register or clock gate is a combinational function of nets in the design. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock is passed to the registers in the fanout of the clock gate. Power Compiler uses the enable condition of the registers for clock-gate insertion.

Enable conditions are represented by Boolean expressions for nets. For example:

```
module TEST (en1, en2, en3, in, clk, dataout);
 input en1, en2, en3, clk;
 input [5:0] in;
 output [5:0] dataout;
 reg [5:0] dataout;

 wire enable;

 assign enable = (en1 | en3) & en2;

 always @(posedge clk) begin
 if(enable)
 dataout <= in;
 else
 dataout <= dataout;
 end

endmodule
```

In this example, the enable condition for the register bank `dataout_reg*` can be expressed as `en1 en2 + en3 en2`.

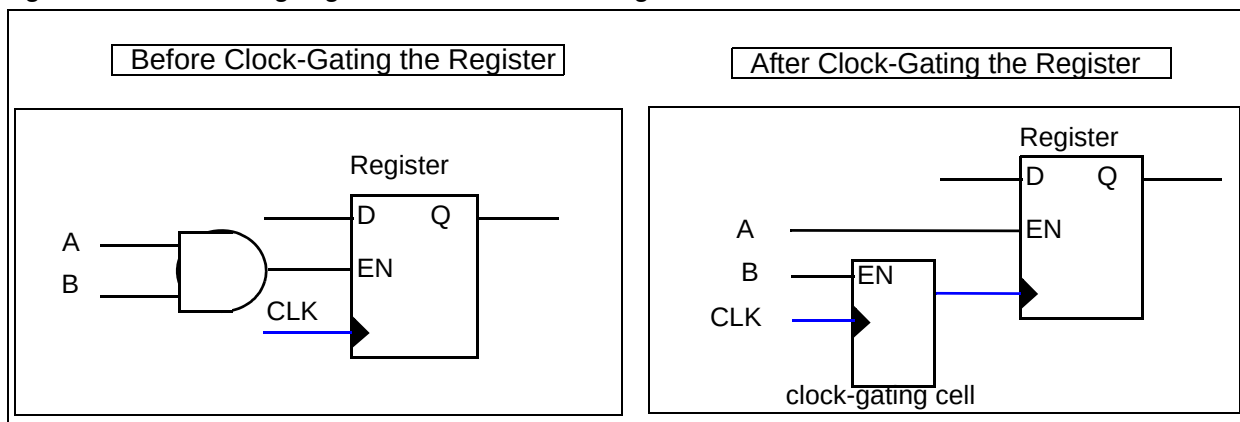


### Excluding Specific Signals From the Enable Condition

You can specify signals to be excluded from the enable condition of clock gating. For example, you can specify a late arriving signal to be excluded from the enable condition, to prevent it from becoming a critical path.

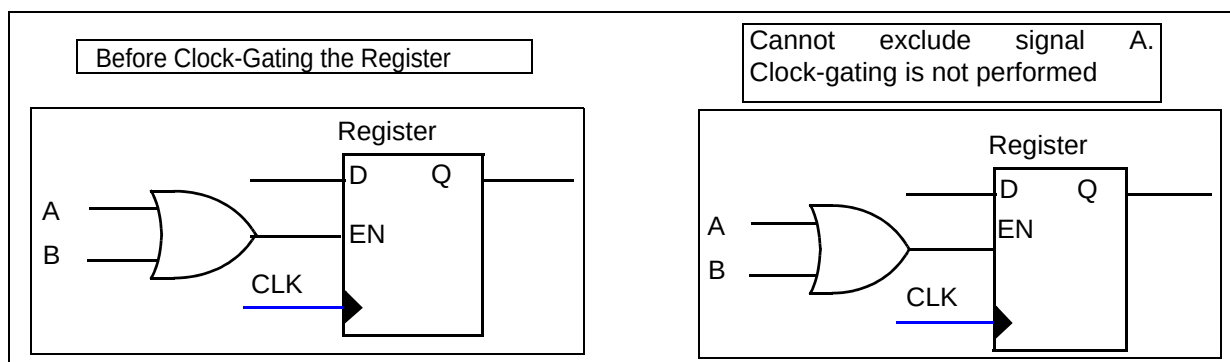
The exclusion of a signal from the enable condition depends on the Boolean expression of the enable condition. In [Figure 7-3](#), the enable signal of the register is an AND function of inputs A and B. To exclude the signal A from the computation of the enable condition of the clock gate, the tool connects input A to the enable pin of the register and input B to the enable pin of the clock-gating cell.

**Figure 7-3** Excluding Signal A from Clock Gating



In [Figure 7-4](#), the enable signal of the register is an OR function of inputs A and B. The tool does not exclude input A from clock-gating because it is not feasible to gate the register when one of the inputs is at logic 1.

**Figure 7-4** Cannot Exclude Signal A. Clock-Gating is not Performed for the Register



You can exclude a signal from the enable expression of a register, if removing the signal from the enable expression does not result in a constant 0 or a constant 1.

Use the `set_clock_gating_enable -exclude` command to specify the objects whose signals are to be excluded from the enable condition. You can specify objects such as primary input ports, output pins of sequential cells, sequential cells, black box cells, and macro cells to be excluded from the enable condition.

The specified signal is excluded from clock-gating when you run the `compile_ultra -gate_clock` command or any subsequent `compile_ultra -incremental -gate_clock` command. Using the exclusion criteria, the `compile_ultra` command checks the feasibility of excluding the specified signals from clock gating. If exclusion is feasible, the command modifies the enable expression of the clock-gating signal and the enable signal of the register.

If it is not feasible to exclude the specified signal from clock-gating, the tool does not clock-gate the register. If the register is already clock-gated using the signal that is specified for exclusion, the tool removes the clock-gating cell. The `set_clock_gating_objects -force_include` command or the `power_cg_all_registers` variable setting does not prevent the tool from removing the clock-gating cell.

Use the `set_clock_gating_enable -undo` command to remove the exclusion constraint.

The `report_clock_gating -ungated` command reports the details of registers that are not clock-gated, reason for not gating, and so on, as shown in [Example 7-8 on page 7-82](#).

The `write_script` command writes out the exclusion constraint that you specify. You can source the file written by the `write_script` command, in the Design Compiler tool to support ASCII flow or in third-party tools.

For more details, see the `set_clock_gating_enable` command man page.

## Setup Condition

To perform clock gating, Power Compiler requires that the enable signal of the register bank is synchronous with its clock. This is the setup condition.

For latch-based or integrated clock gating, Power Compiler can insert clock gating irrespective of the enable signal's and the clock's clock domains. If the enable signal and the register bank reside in different clock domains, you must ensure that the two clock domains are synchronous and that the setup and hold times for the clock-gating cell meet the timing requirements.

For latch-free clock gating, if any of the following characteristics exist, the setup condition is false and the register bank is not gated:

- If the register bank and its controlling logic (including flip-flops) belong to different clock domains, the setup condition is false.
- If the register bank and its controlling logic (including flip-flops) are driven by different edges of the same clock signals, the setup condition is false.

- If the controlling logic is driven by a combinational path from the input port, the setup condition is false, unless:
  - For primary input ports, you specified a clock with the `set_input_delay` command.
  - You specified `power_cg_derive_related_clock true`, which enables clock propagation of the related clocks from parent hierarchies for inputs on subdesigns. The default is false.

These two special cases specify that an input port is synchronous with a given clock; therefore, the setup condition is true.

Specify `power_cg_ignore_setup_condition true` for Power Compiler to ignore the setup condition for latch-free clock gating.

---

## Enabling or Disabling Clock Gating on Design Objects

You can enable or disable clock gating on certain design objects by overriding all necessary conditions set by the clock-gating style. The `set_clock_gating_objects` command specifies the design objects on which clock gating should be enabled or disabled during the `compile_ultra -gate_clock` command. If you use the `insert_clock_gating` command, you must run the `uniquify` command before inserting the clock gates.

The following example includes and excludes the specified registers from clock gating:

```
dc_shell> set_clock_gating_objects \
 -force_include ADDER/out1_reg[*] \
 -exclude ADDER/out2_reg[*]
```

The following example excludes all registers in the subdesign ADDER, except the out1\_reg bank. The out1\_reg bank is clock gated according to the specified clock-gating style:

```
dc_shell> set_clock_gating_objects \
 -exclude ADDER \
 -include ADDER/out1_reg[*]
```

The following example sets and then removes the inclusion and exclusion criteria specified by the `-include` and `-exclude` options:

```
dc_shell> set_clock_gating_objects \
 -include ADDER/out1_reg[*] \
 -exclude ADDER/out2_reg[*]

dc_shell> set_clock_gating_objects \
 -undo {ADDER/out1_reg[*] ADDER/out2_reg[*]}
```

For more information, see the man page.

---

## Inserting Clock Gates

Power Compiler inserts clock-gating cells to your design if you compile your design using the `-gate_clock` option of the `compile` or `compile_ultra` command. You can also insert clock gates to your design using the `insert_clock_gating` command. The following sections discuss in detail these two ways of clock-gate insertion.

---

### Using the `compile_ultra -gate_clock` Command

During the compilation process, Power Compiler can insert clock-gates to your design if you use the `-gate_clock` option of the `compile_ultra` command. With the `-gate_clock` option, the `compile_ultra` command can perform clock-gate insertion on the gate-level netlist, RTL netlist, as well as GTECH netlist. By default, when you use the `-gate_clock` option, the tool inserts clock gates only in the same level of hierarchy as the registers gated by the clock gate. For the tool to perform clock gating across the design hierarchy, set the `compile_clock_gating_through_hierarchy` variable to `true`. For more details about hierarchical clock gating, see [“Hierarchical Clock Gating” on page 7-66](#).

The `compile_ultra -gate_clock` command can also perform clock gating on DesignWare components. For more details, see [“Performing Clock-Gating on DesignWare Components” on page 7-77](#).

In Design Compiler topographical mode, when you perform clock gating by using the `compile_ultra -incremental -gate_clock` command, the tool performs incremental placement and gate-level clock gating.

---

### Using the `insert_clock_gating` Command

The `insert_clock_gating` command can be used to perform clock gating on the GTECH netlist. You cannot use this command to perform clock gating on gate-level netlist. To perform clock gating on a gate-level netlist use the `compile_ultra -gate_clock` command. This command identifies clock-gating opportunities by combining different register banks that share common enable signal.

The `insert_clock_gating` command performs clock gating on all the subdesigns in the design hierarchy by processing each subdesign independently. Use the `-no_hier` option to limit the clock-gate insertion to the top level of the design hierarchy. Use the `-global` option to perform hierarchical clock gating, that is, to insert clock gates on all levels of design hierarchy, considering the design as a whole and not considering each subdesign independently.

For more information about hierarchical clock gating, see [“Hierarchical Clock Gating” on page 7-66](#). For more information about the `insert_clock_gating` command, see the man page.

---

## Clock-Gate Insertion in Multivoltage Designs

In a multivoltage design, the different hierarchies of the design can have different operating condition definition and use different target library subsets. While inserting clock-gating cells in a multivoltage design, Power Compiler chooses the appropriate library cells based on the specified clock-gating style as well as the operating conditions that match the operating conditions of the hierarchical cell of the design. If you do not specify a clock-gating style, the tool chooses a suitable clock-gating style. If the tool does not find a library cell that suites the clock-gating style and the operating conditions, a clock-gating cell is not inserted and a warning message is issued.

For more information about clock-gating style, see [“Selecting Clock-Gating Style” on page 7-26](#).

---

## Clock Gating Flows

The various clock-gating flows supported by the tool are discussed in detail in the following sections.

---

### Inserting Clock Gates in the RTL Design

To insert clock gating logic in your RTL design and to synthesize the design with the clock-gating logic, follow these steps:

1. Read the RTL design.
2. Use the `compile_ultra -gate_clock` command to compile your design.

During the compilation process the clock gate is inserted on the registers qualified for clock-gating. By default, during clock-gate insertion, the `compile_ultra` command uses the default settings of the `set_clock_gating_style` command, and also honors the setup, hold, and other constraints specified in the technology libraries. To override the setup and hold values specified in the library, use the `set_clock_gating_style` command before compiling your design.

You can also use the `insert_clock_gating` command to insert the clock-gating cells.

The `compile_ultra` and `insert_clock_gating` commands use the default settings of the `set_clock_gating_style` command during clock-gate insertion. The default settings of the `set_clock_gating_style` command is suitable for most designs. For

more information about the default clock-gating style, see [“Default Clock-Gating Style” on page 7-24](#).

3. If you are using testability in your design, use the `insert_dft` command to connect the `scan_enable` and the `test_mode` ports or pins of the integrated clock-gating cells.
4. Use the `report_clock_gating` command to report the registers and the clock-gating cells in the design. Use the `report_power` command to get information of the dynamic power used by the design after the clock-gate insertion.

In the following example, clock gating is performed during the compilation process. The default settings of the `set_clock_gating_style` command are used during the clock-gate insertion. The `-scan` option of the `compile_ultra` command enables the examination of your design for scan insertion.

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name CLK
dc_shell> compile_ultra -gate_clock -scan
dc_shell> insert_dft
dc_shell> report_clock_gating
dc_shell> report_power
```

---

## Inserting Clock Gates in Gate-Level Design

To insert clock gating logic in your gate-level netlist and to resynthesize the design with the clock gating logic, follow these steps:

1. Read the gate-level netlist.
2. Use the `compile_ultra -gate_clock -incremental` command to compile your design.
3. During the compilation process, clock-gating cells are inserted on the registers qualified for clock gating. During this process, by default, the `compile_ultra` command
  - Reads the setup and hold constraints that are specified in the technology libraries.
  - Propagates these constraints up the hierarchy.

To override the setup and hold values specified in the library, use the `set_clock_gating_style` command before compiling your design. Use the `compile_ultra -gate_clock` command to perform clock-gate insertion on DesignWare components. For more information about clock-gate insertion on DesignWare components, see [“Performing Clock-Gating on DesignWare Components” on page 7-77](#).

The `compile_ultra -gate_clock` command uses the default settings of the `set_clock_gating_style` command, during the clock-gate insertion. The default settings of the `set_clock_gating_style` command are suitable for most designs. For

more information about the default clock-gating style, see [“Default Clock-Gating Style” on page 7-24](#).

4. If you are using testability in your design, use the `insert_dft` command to connect the `scan_enable` and `test_mode` ports or pins of the integrated clock-gating cells.
5. Use the `report_clock_gating` command to report the registers and the clock gating cells in the design. Use the `report_power` command to get details of the dynamic power used by the design after the clock-gate insertion.

In the following example, clock gating is implemented in the design during the compilation process. The default settings of the `set_clock_gating_style` command are used during the clock-gate insertion.

```
dc_shell> read_ddc design.ddc
dc_shell> compile_ultra -incremental -gate_clock -scan
dc_shell> insert_dft
dc_shell> report_clock_gating
dc_shell> report_power
```

---

## Ensuring Accuracy When Using Ideal Clocks

When using ideal clocks, set the clock transition time to 0 before analyzing the power of your design. To set the clock transition time to 0, use the `set_clock_transition` command.

The presence of clock-gating circuitry leads to a nonzero transition time on the gated clock signal. This increases with the number of flip-flops being gated by the signal. A large transition time at the clock pin of the gated flip-flop leads to a very high internal power usage. However, this is not realistic because the clock tree synthesis tool inserts buffers to reduce clock edge transition time. Setting the clock transition to 0 ensures the most accurate analysis of timing and power after insertion of clock-gating circuitry and before clock tree synthesis.

---

## Specifying Clock-Gate Latency

During synthesis, Design Compiler assumes that the clocks are ideal. An ideal clock incurs no delay through the clock network. This assumption is made because real clock-network delays are not known until after clock tree synthesis. In reality clocks are not ideal and there is a non-zero delay through the clock network. For designs with clock gating, the clock-network delay at the registers is different from the clock-network delay at the clock-gating cell. This difference in the clock-network delay at the registers and at the clock-gating cell results in tighter constraints for the setup condition at the enable input of the clock-gating cell.

For Design Compiler to account for the clock network delays during the timing calculation, specify the clock network latency using either the `set_clock_latency` or the `set_clock_gate_latency` command. The `set_clock_gate_latency` command can be used for both, gate-level and RTL designs.

More details of these commands are described in the following sections.

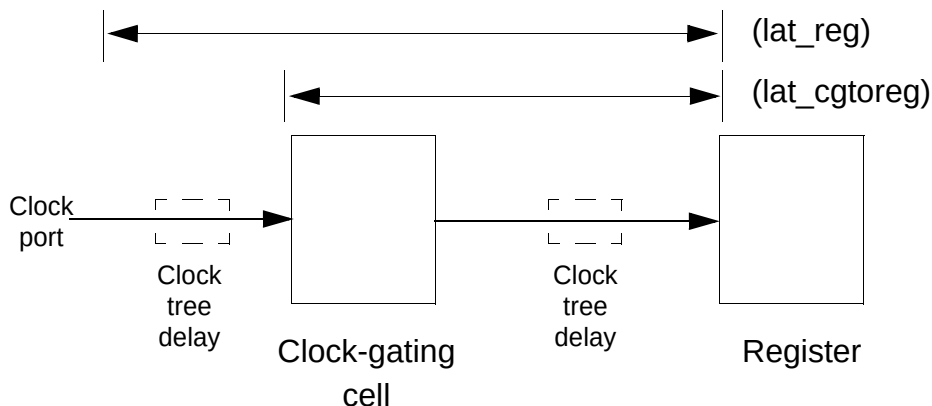
## The `set_clock_latency` Command

Use the `set_clock_latency` command to specify clock network latency for specific clock-gating cells.

In [Figure 7-5](#),

- `lat_cgtoreg` is the estimated delay from the clock pin of the clock-gating cell to the clock pin of the gated register.
- `lat_reg` is the estimated clock-network latency to the clock pins of the registers without clock gating.

*Figure 7-5 Clock Latency With Clock-Gating Design*



For all clock pins of registers (gated or ungated) in the design that are driven by a specific clock, use the `lat_reg` value for the `set_clock_latency` command. For clock pins of all the clock-gating cells, use the value of `lat_reg - lat_cgtoreg` for the `set_clock_latency` command. Because the purpose of setting the latency values is to account for the different clock-network delays between the registers and the clock-gating cell it is important to get a reasonably accurate value of the difference (`lat_cgtoreg`). The absolute values used are relatively less important, unless you are using these values to account for clock-network delay issues not related to clock gating.

For more details, see the command man page.



---

## The `set_clock_gate_latency` Command

When you use the `compile_ultra -gate_clock` command, clock gates are inserted during the compilation process. To specify the clock network latency before the clock-gating cells are inserted by the tool, use the `set_clock_gate_latency` command. This command lets you specify the clock network latency for the clock-gating cells as a function of the clock domain, clock-gating stage, and the fanout of the clock-gating cell. The latency that you specify is annotated on the clock-gating cells when they are inserted by the `compile_ultra -gate_clock` command. You can manually annotate the latency values on the existing clock-gating cells in your design using the `apply_clock_gate_latency` command. For more details, see [“Applying Clock-Gate Latency” on page 7-17](#).

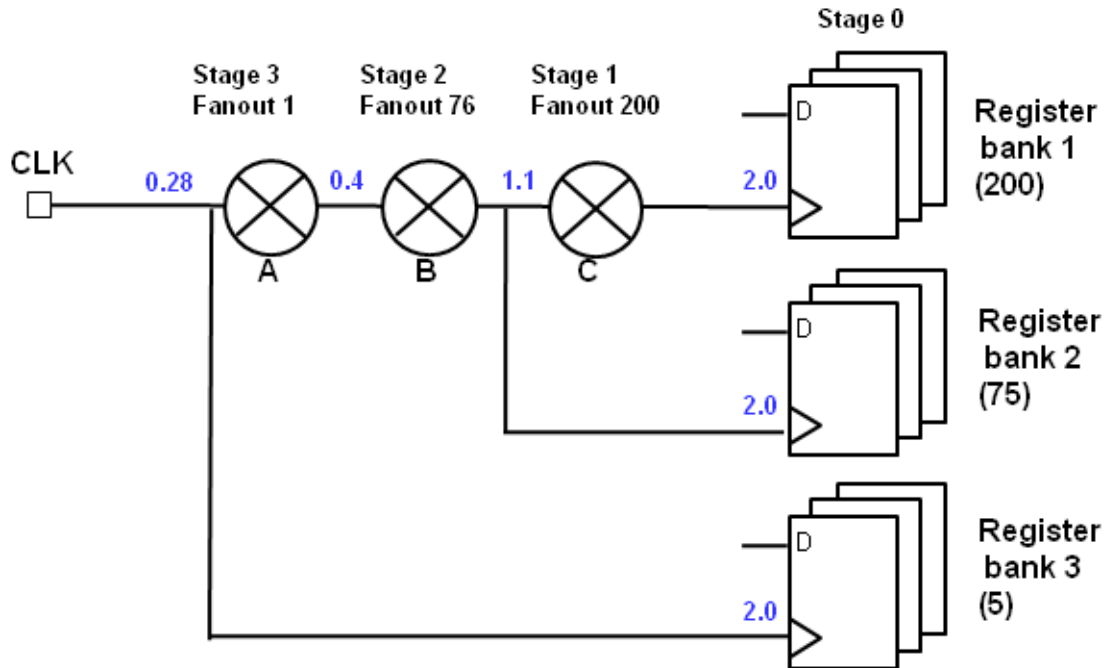
[Figure 7-6](#) shows the definitions for the clock-gate stages and the fanouts.

The clock-gating cell C drives 200 registers. So the fanout of the cell C is 200. Because C drives registers, and not other clock gating cells, the clock gating stage for the cell C is 1.

The clock-gating cell B drives a set of 75 registers and a clock gating cell C. So the fanout of the clock-gating cells B is 76. The clock-gating stage for the cell B is 2; clock gating stage of cell C plus 1.

Similarly, the clock-gating stage of cell A is 3 and the fanout is 1. The clock-gating stage of all the registers is stage 0.

Figure 7-6 Clock-Gating Stages and Fanouts



The following example script shows how to specify the latency values for the various clock gate stages and fanouts using the `set_clock_gate_latency` command for the design shown in Figure 7-6.

```
set_clock_gate_latency -clock CLK -stage 0 \
 -fanout_latency {1-inf 2.0}
set_clock_gate_latency -clock CLK -stage 1 \
 -fanout_latency {1-30 1.8, 31-100 1.5, 101-inf 1.1}
set_clock_gate_latency -clock CLK -stage 2 \
 -fanout_latency {1-5 0.9, 6-20 0.5, 21-100 0.4, 101-inf 0.3}
set_clock_gate_latency -clock CLK -stage 3 \
 -fanout_latency {1-10 0.28, 11-inf 0.11}
```

To specify clock latency value for the clock-gated registers, use the `-stage` option with a value 0. Because you are specifying the latency value for the clock gated registers, the value for the `-fanout_latency` option should be 1-infinity, as shown in the following example:

```
set_clock_gate_latency -clock CLK -stage 0 \
 -fanout_latency {1-inf 1.0}
```

For more information, see the command man page.

---

## Applying Clock-Gate Latency

The clock latency specified using the `set_clock_gate_latency` command is annotated on the registers during the `compile_ultra -gate_clock` command when the clock-gating cells are inserted. However, if you modify the latency values on the clock gates after the compilation, you must manually apply the latency values on the existing clock-gating cells using the `apply_clock_gate_latency` command.

### Note:

After you modify the clock-gate latency using the `set_clock_gate_latency` command, if you compile your design using the `compile_ultra` or `compile_ultra -incremental` command, it is not necessary to use the `apply_clock_gate_latency` command to apply the latency values. The tool annotates the specified value during compilation.

For more details, see the command man pages.

---

## Resetting Clock-Gate Latency

To remove the clock latency information specified on the clock-gating cells, use the `reset_clock_gate_latency` command. This command removes the clock latency values on the specified clocks. If you do not specify the clock, the clock latency values on all the clock-gating cells are removed. This command removes the clock latency on the specified clocks, irrespective of whether the latency values were specified using the `set_clock_latency` or `set_clock_gate_latency` command.

For more details, see the command man page.

---

## Comparison of the Clock-Gate Latency Specification Commands

[Table 7-1](#) compares various commands that you can use to specify the clock-gate latency.

*Table 7-1 Comparison of Clock-Gating Latency Specification Commands*

| <code>set_clock_gate_latency</code>                                         | <code>set_clock_gating_style -setup -hold</code>                                                                                  | <code>set_clock_gating_check</code>                               | <code>set_clock_latency</code>                               |
|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|--------------------------------------------------------------|
| Recommended for use with the <code>compile_ultra -gate_clock</code> command | Default settings are recommended for most designs. Use this command only if the default settings are not suitable for your design | To specify the clock-gate latency on existing clock-gating cells. | To modify clock-gate latency on existing clock-gating cells. |

*Table 7-1 Comparison of Clock-Gating Latency Specification Commands (Continued)*

| <b>set_clock_gate_latency</b>                                                                                           | <b>set_clock_gating_style -setup -hold</b>                                   | <b>set_clock_gating_check</b>                                            | <b>set_clock_latency</b>                                                      |
|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| To specify clock-gate latency before the clock gates are inserted by the <code>compile_ultra -gate_clock</code> command |                                                                              | Specification is on the instance. So, specify on each clock-gating cell. | Specification is on the instance. So, specify on each clock-gating cell       |
| To modify the clock-gate latency settings on existing clock-gating cells                                                | To specify the setup and hold values before the clock gates are inserted     | Specification overrides the setup and hold values in the library         | The latency setting specifies the clock arrival time at the clock-gating cell |
| The latency setting specifies the clock arrival time at the clock-gating cell                                           | The specification overrides the setup and hold values defined in the library |                                                                          |                                                                               |
| Specification is based on clock domain, clock-gating state and fanout                                                   | Generic settings for all the clock gates in the design                       |                                                                          |                                                                               |

## Calculating the Clock Tree Delay From Clock-Gating Cell to Registers

If your clock tree synthesis tool does not insert buffers after the clock-gating cell, then the total delay between the clock-gating cell and the registers is equal to the delay of the clock-gating cell (clock pin to clock out signal) plus the wire delay between the clock-gating cell and the registers. If your clock tree synthesis tool inserts buffers after the clock-gating cell, add an estimate of the clock-network delay to the total delay between the clock-gating cell and the registers. You can use an estimate based on the fanout of the clock-gating cell and the driving capacity of typical clock tree buffers or use data from earlier designs.

For most designs, the enable signal arrives early and is not affected by clock-network delay issues. For late arriving enable signals, it is advised to be conservative (high value) in the selection of the delay from the clock-gating cell to the registers. A low value might mean an enable signal which is unable to meet arrival time constraints at the clock-gating cell after the clock tree is inserted. However, a high value might over constrain the enable signal

leading to higher area or power and ensures that the enable signal arrives in time at the clock-gating cell.

After placement and clock tree synthesis, you can back-annotate delay information by using the `set_propagated_clock` command for Design Compiler to use real delay data for the clock-network delay. For more information, see the *Design Compiler User Guide*.

---

## Specifying Setup and Hold

During insertion of clock gates, the setup and hold time that you specify defines the margins within which the enable signal (EN) must operate to maintain the integrity of the gated-clock signal.

The setup and hold values for the integrated clock-gating cell are specified in the logic library. The values specified in the logic library are honored by the `compile_ultra -gate_clock` command during the clock-gate insertion. However, you can override these values in the following ways:

- Specifying the `-setup` and `-hold` options of the `set_clock_gating_style` command. By doing so, all the clock gates in the design should have the setup and hold time that you specify.
- For the clock-gating cells already existing in your design, use the `set_clock_gating_check` command to specify a desired setup and hold time.

You use the `report_timing -to` command to the enable pin of the clock-gating cell to verify that the new values are correct.

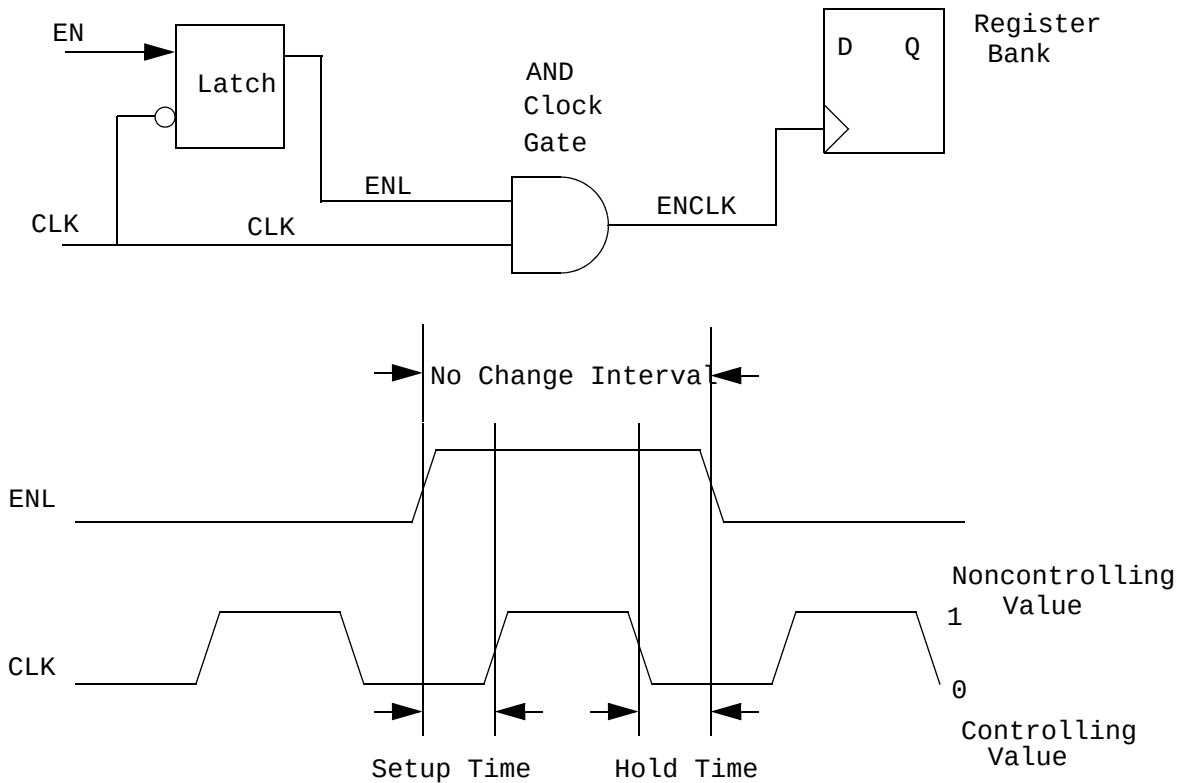
The following example uses the `set_clock_gating_style` command to specify the setup and hold values:

```
set_clock_gating_style \
 -max_fanout 16 -positive_edge_logic integrated \
 -setup 6 -hold 2
compile_ultra -gate_clock
to validate the user-specified setup or hold time for
integrated clock gating
report_timing -to clk_gate_out_top_reg/EN
report_timing -to clk_gate_out_top_reg_1/EN
```

The clock gate must not alter the waveform of the clock, other than turning the clock signal on and off. If the enable signal operates outside the chosen margins specified by the `-setup` and `-hold` options, the resulting gated signal might be clipped or corrupted.

[Figure 7-7](#) and [Figure 7-8 on page 7-21](#) show the relationship of setup and hold time to a clock waveform. [Figure 7-7](#) shows the relationship with an AND gate as the clock-gating element. [Figure 7-8 on page 7-21](#) shows the relationship with an OR gate as the clock-gating element.

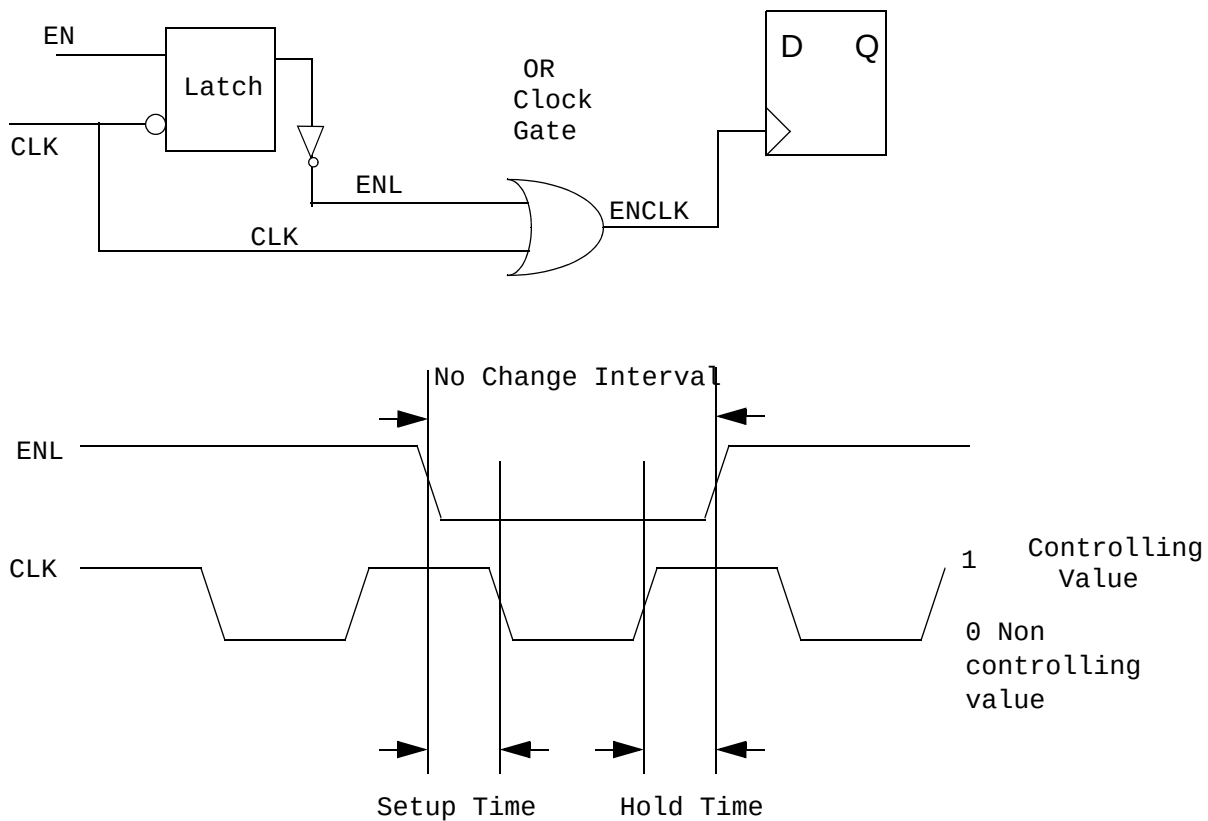
Figure 7-7 Setup and Hold Time for an AND Clock Gate



Enable after latch (ENL) signal must be stable before the clock input (CLK) makes a transition to a non-controlling value. The hold time ensures that the ENL is stable for the time you specify after the CLK returns to a controlling value. The setup and hold time ensures that the ENL signal is stable for the entire time that the CLK signal has a non-controlling value, which prevents clipping or glitching of the ENCLK clock signal.

You might need to add latency by using the `set_clock_latency` command. Use this command for non-clock-gating registers. For more information, see [“Specifying Clock-Gate Latency” on page 7-13](#) and the Design Compiler documentation.

Figure 7-8 Setup and Hold Time for an OR Clock Gate

**Note:**

When using PrimeTime for static timing analysis, use the `-setup` and `-hold` options of the `set_clock_gating_check` command to change the setup and hold values for the gating check. PrimeTime performs clock-gating checks on all gated clocks using 0.0 as the default for setup and hold.

---

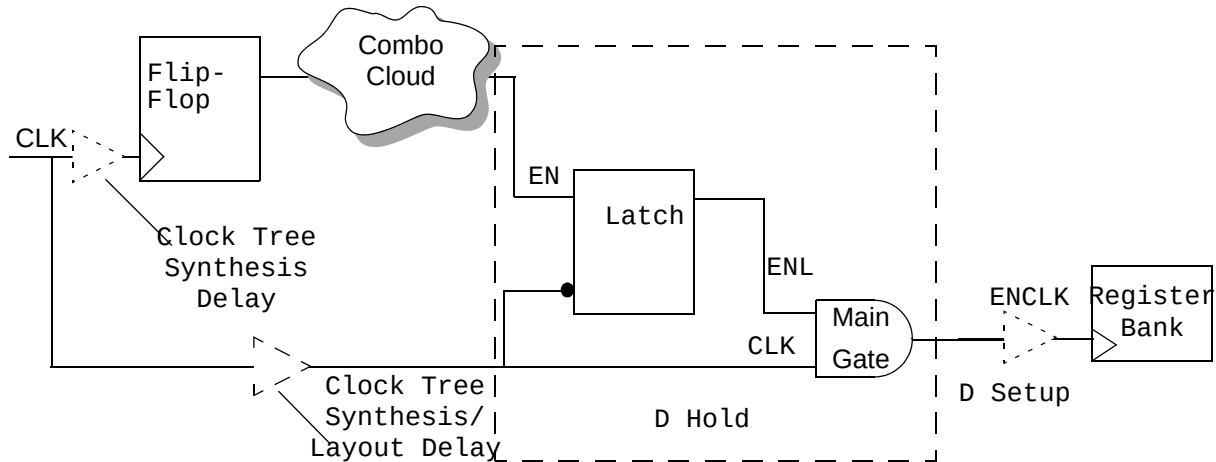
## Predicting the Impact of Clock Tree Synthesis

Clock tree synthesis can affect your choice of setup and hold time. However, during clock gating, the clock tree does not exist yet: clock tree synthesis normally occurs much later in the design process than clock gating. Without the clock tree, it can be difficult to precisely predict the impact of clock tree synthesis on the delay of the design. For this reason, you might find it necessary to alter your setup and hold time after clock tree synthesis.

## Choosing a Value for Setup

Choose a value for the setup time that estimates the impact of the delay of the clock tree from the clock gate to the gated register bank. In latch-based clock gating, the value for setup mimics the delay of the clock tree from the clock gate to the register bank.

Figure 7-9 Setup and Hold Time for Clock Tree Synthesis



Your setup time constrains the ENL signal so that after gate-level synthesis, there is still enough timing slack for the addition of the clock tree during clock tree synthesis.

In latch-free clock gating, the value for setup must consider the clock signal duty cycle. For example, in a design using a latch-free clock gate:

1. Estimate the delay of the clock tree between the clock gate and the gated register (as you would for the latch-based clock gate).
2. From the value you estimate in step 1, add the worst-case (largest possible) clock low time (typically half of the clock-cycle time).

This is appropriate for flip-flops triggered on the clock's rising edge. For flip-flops triggered on the clock's falling edge, add the worst-case (largest possible) clock high time.

If the setup time is too small, the ENL signal must be reoptimized after back-annotation from layout to fit the tighter timing constraints. If the value of -setup is too large, the ENL signal is too constrained and optimization of combinational control logic results in larger area and power to satisfy the tighter timing constraints.



---

## Choosing a Value for Hold

Latch-based clock gating has the timing requirement that the transition of the ENL signal occur at the 2-input clock gate after the trailing edge (rising edge for falling-edge flip-flop) of the clock signal. This timing requirement is usually satisfied because the addition of a latch because of clock gating, increases the delay on the ENL signal. In rare cases, however, after clock tree synthesis and physical design, additional delay in the clock signal might cause the CLK signal to arrive after the ENL signal. This is due to clock skew between the clock signal driving the clock-gating latch and the clock signal driving the 2-input gate.

If you expect this timing violation, you can set the `-hold` value during clock gating to artificially define a hold constraint on the ENL signal. Gate-level synthesis adds buffers in the ENL signal if they are necessary to satisfy your hold constraint.

If the value of `-hold` is too small, you might have to reoptimize the ENL signal after back-annotation from layout to ensure the integrity of the gated clock signal. If the value of `-hold` is too large, you might find a chain of buffers delaying the ENL signal before the clock gate.

---

## Clock-Gating Styles

Power Compiler inserts the clock-gating cells in the design based on the styles that you specify. When you do not specify a clock-gating style, the tool uses a set of predefined styles for the clock gates. The default settings of the `set_clock_gating_style` command are suitable for most designs.

The following sections discuss in detail, the default clock-gating style and using specific clock-gating styles:

- [Default Clock-Gating Style](#)
- [Selecting Clock-Gating Style](#)

The `compile_ultra -gate_clock` command prevents clock-gate insertion when the target library does not contain cells for the defined clock-gating style and operating condition and issues the PWR-763 information message. You must redefine the clock-gating style or the operating conditions, based on the clock-gating cells available in the target library.

---

## Default Clock-Gating Style

When you specify the `set_clock_gating_style` command, the default style used by the tool is different from the default style used when you do not specify the command.

When you specify the `set_clock_gating_style` command with only a few options, the tool uses the default specified in [Table 7-2](#) for the unspecified option.

*Table 7-2 Defaults for Clock-Gating Style*

| Parameter               | Default values used when the <code>set_clock_gating_style</code> command is specified without any option |
|-------------------------|----------------------------------------------------------------------------------------------------------|
| Sequential cell         | Latch                                                                                                    |
| Minimum bit-width       | 3                                                                                                        |
| Setup constraint        | 0                                                                                                        |
| Hold constraint         | 0                                                                                                        |
| Positive edge logic     | and                                                                                                      |
| Negative edge logic     | or                                                                                                       |
| Control point           | none                                                                                                     |
| Control signal          | scan_enable                                                                                              |
| Observation point       | false                                                                                                    |
| Observation logic depth | 5                                                                                                        |
| Maximum fanout          | infinite                                                                                                 |
| Number of stages        | 1                                                                                                        |
| No sharing              | false                                                                                                    |

When you specify the `set_clock_gating_style` command multiple times, the last setting overrides the previous settings.

When you do not specify a clock-gating style, Power Compiler derives a default clock-gating style based on the specified libraries. The cells are chosen from the library in the following decreasing order of priority:

- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic integrated \`  
`-control_point before -control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic integrated -control_point after \`  
`-control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic integrated -control_point before \`  
`-control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic integrated -control_point after \`  
`-control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic integrated`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic or -control_point after \`  
`-control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic or -control_point before \`  
`-control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic or -control_point after \`  
`-control_signal test_mode`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic or -control_point after \`  
`-control_signal test_mode -observation_point true`
- `set_clock_gating_style -positive_edge_logic integrated \`  
`-negative_edge_logic or`
- `set_clock_gating_style -positive_edge_logic and \`  
`-negative_edge_logic integrated -control_point before \`  
`-control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic and \`  
`-negative_edge_logic integrated -control_point after \`  
`-control_signal scan_enable`
- `set_clock_gating_style -positive_edge_logic and \`

- ```
-negative_edge_logic integrated -control_point before \
-control_signal test_mode -observation_point true\
```
- `set_clock_gating_style -positive_edge_logic and \`
`-negative_edge_logic integrated -control_point after \`
`-control_signal test_mode -observation_point true`
 - `set_clock_gating_style -positive_edge_logic and \`
`-negative_edge_logic integrated`
 - `set_clock_gating_style -positive_edge_logic and \`
`-negative_edge_logic or`

The following example inserts clock-gating cells by choosing a suitable default style:

```
read_verilog low.v
compile_ultra -gate_clock
report_clock_gating -style
compile_ultra -incremental
```

Selecting Clock-Gating Style

Use the `set_clock_gating_style` command to select the clock-gating style. The `compile_ultra -gate_clock` and the `insert_clock_gating` commands use the specified clock-gating style to insert the clock-gating cells. The default settings of the `set_clock_gating_style` command is suitable for most designs. If the default setting does not suit your design, use the `set_clock_gating_style` command to change the default setting.

The clock-gating style that you specify is applied to the entire design. You can also apply the clock-gating style only to specific power domains or hierarchical cells of the design. For more information about specifying clock-gating styles on specific instances, see [“Using Instance-Specific Clock-Gating Style” on page 7-43](#).

The `set_clock_gating_style` command has the following syntax:

```
set_clock_gating_style
[-sequential_cell none | latch]
[-minimum_bitwidth int]
[-setup sh_value]
[-hold sh_value]
[-positive_edge_logic {cell_list | \
integrated [active_low_enable][invert_gclk]]]
[-negative_edge_logic {cell_list | \
integrated [active_low_enable][invert_gclk]]]
[-control_point before | after]
[-control_signal scan_enable | test_mode]
[-observation_point true | false]
[-observation_logic_depth int]
[-max_fanout int]
```

```
[ -num_stages int ]  
[ -no_sharing ]  
[ -instances instances ]  
[ -power_domains power_domains ]
```

The following sections describe how to use the `set_clock_gating_style` command:

- [Choosing Gating Logic](#)
- [Choosing an Integrated Clock-Gating Cell](#)
- [Choosing a Configuration for Discrete Gating Logic](#)
- [Choosing a Simple Gating Cell by Name](#)
- [Choosing a Simple Gating Cell and Library by Name](#)
- [Designating Simple Cells Exclusively for Clock Gating](#)
- [Choosing a Specific Latch and Library](#)
- [Choosing a Latch-Free Style](#)
- [Improving Testability](#)
- [Connecting the Test Ports Throughout the Hierarchy](#)
- [Using Instance-Specific Clock-Gating Style](#)

Choosing Gating Logic

The following options of the `set_clock_gating_style` command specify the type of clock-gating logic or clock-gating cell used for implementing clock gating:

```
-positive_edge_logic [gate_list]  
-negative_edge_logic [gate_list]
```

You can specify a configuration of 1-input and 2-input gates (simple gating cells) to use for clock gating, or an integrated clock-gating cell already defined in the target library. An integrated cell is a dedicated clock-gating cell that combines all of the simple gating logic of a clock gate into one fully characterized cell, possibly with additional logic such as multiple enable inputs, active-low enabling logic, or an inverted gated clock output.

Choosing an Integrated Clock-Gating Cell

You can use the `-positive_edge_logic` and `-negative_edge_logic` options of the `set_clock_gating_style` command to specify the integrated clock-gating cell for clock gating:

```
-positive_edge_logic [gate_list]  
-negative_edge_logic [gate_list]
```

The first cell found that meets the clock-gating requirements is used and possibly sized up or down to meet the design rule violations if the library has integrated cells of different sizes. Use the `power_do_not_size_icg_cells` variable to prevent this behavior.

Choosing an Integrated Cell by Functionality

When selecting an integrated cell by functionality, clock gating searches your library for integrated cells having the correct value of the `clock_gating_integrated_cell` attribute.

Use the `set_clock_gating_style` command to specify the functionality of the integrated cell you want clock gating to look for.

Power Compiler uses the first integrated cell it finds in your library that matches the requirements you specify with the `set_clock_gating_style` command. For example, if you enter

```
set_clock_gating_style -negative_edge_logic {integrated}
```

Power Compiler uses the first integrated cell it finds in your logic library that has the `clock_gating_integrated_cell` attribute, as follows:

```
clock_gating_integrated_cell : "latch_negedge";
```

When you do not specify the sequential option, the tool uses the default latch-based gating. For more information about attributes for integrated cells and library syntax, see the Library Compiler documentation.

Choosing an Integrated Cell by Name

Choose an integrated cell by name when you require a specific integrated cell or if you have more than one integrated cell with the same `clock_gating_integrated_cell` attribute. For example,

```
set_clock_gating_style -positive_edge_logic {integrated:my_cell}
```

In this example, clock gating chooses an integrated cell called `my_cell` from the logic library. For more information about attributes for integrated cells and Library syntax, see the Library Compiler documentation.

Specifying a Subset of Integrated Clock Gates

Use the `set_dont_use -power` command to limit clock gate insertion to a specific set of integrated clock gate cells from one or more libraries. This command guarantees that the specified cells is not used for power optimization. For example,

```
set_dont_use -power [get_lib_cell a1.db/icg_a1_*]
set_dont_use -power [get_lib_cell b2.db/icg_b2_*]
set_dont_use -power [get_lib_cell c3.db/icg_c3_*]
set_clock_gating_style -positive_edge_logic {integrated}
compile_ultra -gate_clock
```

In this example, the `set_clock_gating_style` command directs the `compile_ultra -gate_clock` command to use all integrated cells except the cells that have the `dont_use` attribute.

Using Setup and Hold for Integrated Cells

Setup and hold constraints are built into the integrated cell when you create it with Library Compiler, but you can override the values by using either the `set_clock_gating_style` command or the `set_clock_gating_check` command.

If you provide `-setup` and `-hold` values on the command line when using an integrated cell, the values are overridden.

The following example uses an integrated cell to gate rising-edge-triggered registers and uses simple cells to gate falling-edge-triggered registers using latch-free style.

```
set_clock_gating_style -sequential_cell none
-setup setup_value
-hold hold_value
-positive_edge_logic {integrated}
-negative_edge_logic {inv nor buf}
```

The `setup_value` and `hold_value` apply not only to the integrated cell, but also to the clock gate built for falling-edge-triggered registers using simple cells (INV, NOR, and BUF gates in this example). For more information about integrated clock-gating cells and timing, see the *Library Compiler User Guide*.

Choosing a Configuration for Discrete Gating Logic

The `-positive_edge_logic` and `-negative_edge_logic` options can have up to three string parameters that specify the type of clock gating logic:

- The type of 2-input clock gate (AND, NAND, OR, NOR)
- An inverter or buffer on the clock network before the 2-input clock gate
- An inverter or buffer on the clock network after the 2-input clock gate

The positions of the string parameters determine whether clock gating places a buffer or inverter before or after the 2-input clock gate. For example, if the value of `-positive_edge_logic` is `{and buf}`, clock gating uses an AND gate and places a buffer in the fanout from the AND gate. If the value is `{inv nor}`, clock gating uses a NOR gate and places an inverter in the fanin of the NOR gate. Both of these examples result in AND functionality of the clock gate.

The type of logic that is appropriate for gating your circuit depends on,

- Whether the gated register banks are inferred by rising- or falling-edge clock constructs in your HDL code
and
- Whether you use latch-based or latch-free clock gating

When using latch-free clock gating, you must specify both the `-positive_edge_logic` and `-negative_edge_logic` options.

For proper operation of the gated design, use the `-positive_edge_logic` and `-negative_edge_logic` options of the `set_clock_gating_style` command to choose any combination of gates that provide the appropriate functionality shown in [Table 7-3](#) and [Table 7-4 on page 7-32](#). [Table 7-3](#) provides information for the latch-based clock-gating style.

Table 7-3 Gating Functionality for Latch-Based Clock Gating

Gating logic -pos{} or -neg{}	Latch-based clock gating			
	Rising-edge-triggered registers ¹		Falling-edge-triggered registers ²	
	Valid?	Remarks	Valid?	Remarks
{and}	Yes			
{or}			Yes	(³)
{nand}	Yes	Clock gating adds an inverter to the clock line to the register.		
{nor}			Yes	Clock gating removes the inverter from the clock line to the register.
{and inv}	Yes	Clock gating adds an inverter to the clock line to the register.		
{or inv}			Yes	Clock gating removes the inverter from the clock line to the register.
{nand inv}	Yes			

Table 7-3 Gating Functionality for Latch-Based Clock Gating (Continued)

Gating logic -pos{} or -neg{}	Latch-based clock gating			
	Rising-edge-triggered registers ¹		Falling-edge-triggered registers ²	
	Valid?	Remarks	Valid?	Remarks
{nor inv}			Yes	
{inv and}			Yes	Clock gating removes the inverter from the clock line to the register.
{inv or}	Yes	Clock gating adds an inverter to the clock line to the register.		
{inv nand}			Yes	(⁴)
{inv nor}	Yes			
{inv and inv}			Yes	(⁴)
{inv or inv}	Yes			
{inv nand inv}			Yes	Clock gating removes the inverter from the clock line to the register.
{inv nor inv}	Yes	Clock gating adds an inverter to the clock line to the register.		

1. If Power Compiler adds an inverter on the clock line to a rising-edge-triggered register, Design Compiler might infer a falling-edge-triggered register during later synthesis if one is available in your library. This is normal.

2. If Power Compiler removes an inverter from the clock line to a falling-edge-triggered register, Design Compiler might infer a rising-edge-triggered register if one is available in your library. This is normal.

3. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.

4. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.

Table 7-4 provides information for the latch-free clock-gating style.

Table 7-4 Gating Functionality for Latch-Free Clock Gating

Gating logic -pos{} or -neg{}	Latch-free clock gating			
	Rising-edge-triggered registers ¹		Falling-edge-triggered registers ²	
	Valid?	Remarks	Valid?	Remarks
{and}			Yes	
{or}	Yes	(³)		
{nand}			Yes	Clock gating removes the inverter from the clock line to the register.
{nor}	Yes	Clock gating adds an inverter to the clock line to the register.		
{and inv}			Yes	Clock gating removes the inverter from the clock line to the register.
{or inv}	Yes	Clock gating adds an inverter to the clock line to the register.		
{nand inv}			Yes	
{nor inv}	Yes	(³)		
{inv and}	Yes	Clock gating adds an inverter to the clock line to the register.		
{inv or}			Yes	Clock gating removes the inverter from the clock line to the register.
{inv nand}	Yes	(⁴)		
{inv nor}			Yes	

Table 7-4 Gating Functionality for Latch-Free Clock Gating (Continued)

Gating logic -pos{} or -neg{}	Latch-free clock gating			
	Rising-edge-triggered registers ¹		Falling-edge-triggered registers ²	
	Valid?	Remarks	Valid?	Remarks
{inv and inv}	Yes	(⁴)		
{inv or inv}			Yes	
{inv nand inv}	Yes	Clock gating adds an inverter to the clock line to the register.		
{inv nor inv}			Yes	Clock gating removes the inverter from the clock line to the register.

1. If Power Compiler adds an inverter on the clock line to a rising-edge-triggered register, Design Compiler might infer a falling-edge-triggered register during later synthesis if one is available in your logic library. This is normal.

2. If Power Compiler removes an inverter from the clock line to a falling-edge-triggered register, Design Compiler might infer a rising-edge-triggered register if one is available in your library. This is normal.

3. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.

4. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating. This cancels the effect of the additional inverter on the enable input signal. Therefore only the clock pin of the main gate is inverted.

For example, to achieve AND functionality, you can simply use an AND gate. However, AND functionality also results from the combination of an INV and a NOR gate. Any combination of individual gates is allowable if the combination results in the appropriate functionality shown in [Table 7-3 on page 7-30](#) and [Table 7-4](#).

In the following example, latch-based clock gating uses an AND gate for gating clocks of rising-edge-triggered register banks and an OR gate for gating clocks of falling-edge-triggered register banks. The enable input of the OR gate has an inverter to ensure correct functionality when using clock gating.

```
-positive_edge_logic {and} -negative_edge_logic {or}
```

In the following example, latch-based clock gating chooses a NOR gate for gating clocks of rising-edge-triggered register banks. Clock gating inserts an inverter in the fanin to the

2-input clock gate and a buffer in the fanout from the 2-input clock gate. This combination results in AND functionality.

```
-positive_edge_logic {inv nor buf} -negative_edge_logic {inv and inv}
```

For falling-edge-triggered register banks in this example, clock gating uses an AND gate to gate the clock. Clock gating inserts inverters in the fanin and fanout of the 2-input clock gate. This combination results in OR functionality. The enable input of the OR gate already has an inverter. This cancels the effect of the additional inverter on the enable input signal. Therefore, only the clock pin of the main gate is inverted.

Choosing a Simple Gating Cell by Name

The syntax of the `-positive_edge_logic` and `-negative_edge_logic` options allows you to use a specific clock-gating cell during clock gating. To use a specific gating cell from the target library, specify the cell name after the element type, separated by a colon.

In the following example for rising-edge-triggered register banks, latch-based clock gating chooses the specific AND gate, MYAND2, from the target library. In this example, clock gating inserts a buffer in the fanout of the clock gate.

```
-positive_edge_logic {and:MYAND2 buf}
```

Choosing a Simple Gating Cell and Library by Name

In some cases, you might have more than one target library with cell names that are the same. In such cases, you can use a specific cell from a specific library for clock gating. The syntax of `-positive_edge_logic` and `-negative_edge_logic` allows you to indicate a specific library and cell for clock gating, as follows.

```
target_library = { "CMOS8_MAX.db" "tech_lib1.db" "tech_lib2.db" }
```

```
-positive_edge_logic {and:tech_lib1/MYAND2 buf:tech_lib2/ MYBUF2}
```

In this example, clock gating uses a particular AND cell and BUF cell from different technology libraries. The AND cell is MYAND2 from the tech_lib1 library, and the buffer is MYBUF2 from the tech_lib2 library. You must have previously specified these technology libraries as target libraries by setting the Design Compiler `target_library` variable.

Designating Simple Cells Exclusively for Clock Gating

During technology mapping, Design Compiler builds clock-gating logic, using the generic representation created by Power Compiler and cells from your library.

Unless you are using an integrated cell for gating, there is nothing to prevent Design Compiler from using the same cells for mapping other parts of the design.

You can designate certain cells to be used exclusively or preferentially for gating clocks. Such cells can be the 2-input clock gate, inverters, buffers, or latches used in the latch-based style of clock gating.

To use a specific cell for clock gating and preclude its use in other areas of the design, set the following Library Compiler attributes to `true` in the library description of the cell:

- `dont_use`

When set to `true`, this attribute prevents Design Compiler from choosing the cell when mapping the design to technology.

- `is_clock_gating_cell`

This is an attribute of type Boolean for the cell group. When set to `true`, this attribute identifies the cell for use in clock gating. If `dont_use` and `is_clock_gating_cell` are both set to `true`, the cell is used only in clock-gating circuitry.

You can set `dont_use` and `is_clock_gating_cell` on

- 2-input clock gates

Examples of 2 clock gates are AND, NAND, OR, and NOR library cells that are used to gate clocks.

- 1-input clock gates

Examples of 1 clock gates are buffer and inverter library cells that are used in the fanin and fanout of the 2 clock gate.

- 2-input D latches

These latches can be active high or low and must have a noninverting output.

To use a cell preferentially in clock gating, set only the `is_clock_gating_cell` attribute to `true`. Clock gating uses such cells preferentially when inserting clock-gating circuitry. Later, Design Compiler can use them as well when mapping other parts of the design to the target technology.

For more information about the syntax and use of Library Compiler attributes, see the Library Compiler documentation.

The 2-input clock gate has an enabling input and a clock input that is connected to ENL and CLK signals in [Figure 7-2 on page 7-4](#). If the clock attribute is set on one of the pins of the 2-input clock gate, Power Compiler recognizes the remaining input pin as the enable pin. However, library cell syntax allows you to explicitly designate an input pin as the enabling input. In the pin group of the library description for the cell, set the `clock_gate_enable_pin` attribute to `true`. This is an attribute of type Boolean for the pin group.

Example

```
clock_gate_enable_pin : true;
```

If Power Compiler finds neither a clock attribute nor a `clock_gate_enable_pin` attribute, the software checks for the existence of setup and hold time on the pins. If setup and hold time are found on a pin, the software uses that pin as the enable pin. For more information about Library Compiler syntax and cell descriptions, see the Library Compiler documentation.

Choosing a Specific Latch and Library

The `-sequential_cell` option of `set_clock_gating_style` command allows you to select a clock-gating style that uses latches or avoids the use of latches. [Figure 7-2 on page 7-4](#), earlier in this chapter, shows an example of the latch-based clock-gating style. An example of a circuit with the latch-free clock-gating style is shown in [Figure 7-10](#).

The `-sequential_cell` option allows you to use a specific latch when inserting clock-gating circuitry. To use a specific latch from the target library, specify the name of the latch after the element type, separating the two with a colon (:). For example:

```
-sequential_cell latch:LAH10
```

To designate a specific latch from a specific target library, insert the name of the library as shown in the following example. Clock gating uses a latch called LAH10 from the target library.

In the following example, clock gating uses the LAH10 latch from the SPECIFIC_TECHLIB library.

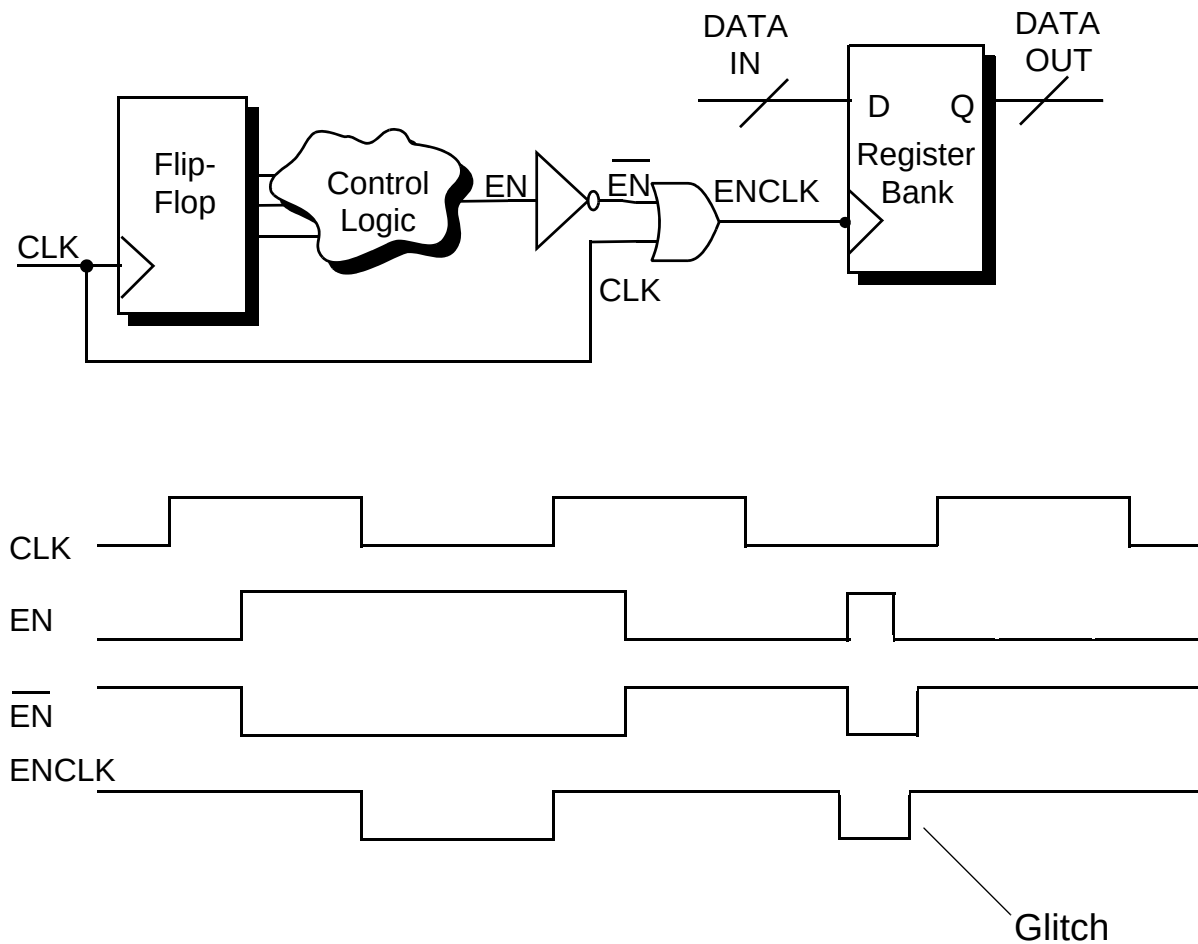
```
-sequential_cell latch:SPECIFIC_TECHLIB/LAH10
```

Choosing a Latch-Free Style

To specify a latch free clock gating style, use the `-sequential_cell none` option of the `set_clock_gating_style` command. For example, in the latch-free style in [Figure 7-10](#), clock pulses to the register bank are gated by the OR gate and it prevents the trailing clock edge. A latch-free clock gate for rising-edge-triggered logic prevents the falling clock edge.

Eliminating the latch can reduce power dissipation and area slightly. However, the latch-free method has a significant drawback: The EN signal must be stable at its new value before the falling clock edge. If the EN signal is not stable before the falling clock edge, glitches on the EN signal can corrupt the clock signal to the register. Any glitches on the EN signal after the trailing edge of the clock lead to glitching and corruption of the gated clock signal. See [Figure 7-10 on page 7-37](#) for an example of latch-free clock gating.

Figure 7-10 Latch-Free Clock Gating



Improving Testability

Clock gating introduces multiple clock domains in the design. Introducing multiple clock domains can affect the testability of your design unless you add logic to enhance testability.

In certain scan register styles, a gated register cannot be included in a scan chain, because gating the register's clock makes it uncontrollable for test (assuming there is no dedicated scan clock). Without the register in the scan chain, test controllability is reduced at the register output and test observability is reduced at the register input. If you have many gated registers, this can significantly reduce the fault coverage in your design.

You can improve the testability of your circuit by using the options of the `set_clock_gating_style` command to determine the amount and type of testability logic added during clock gating. Follow these steps to improve testability:

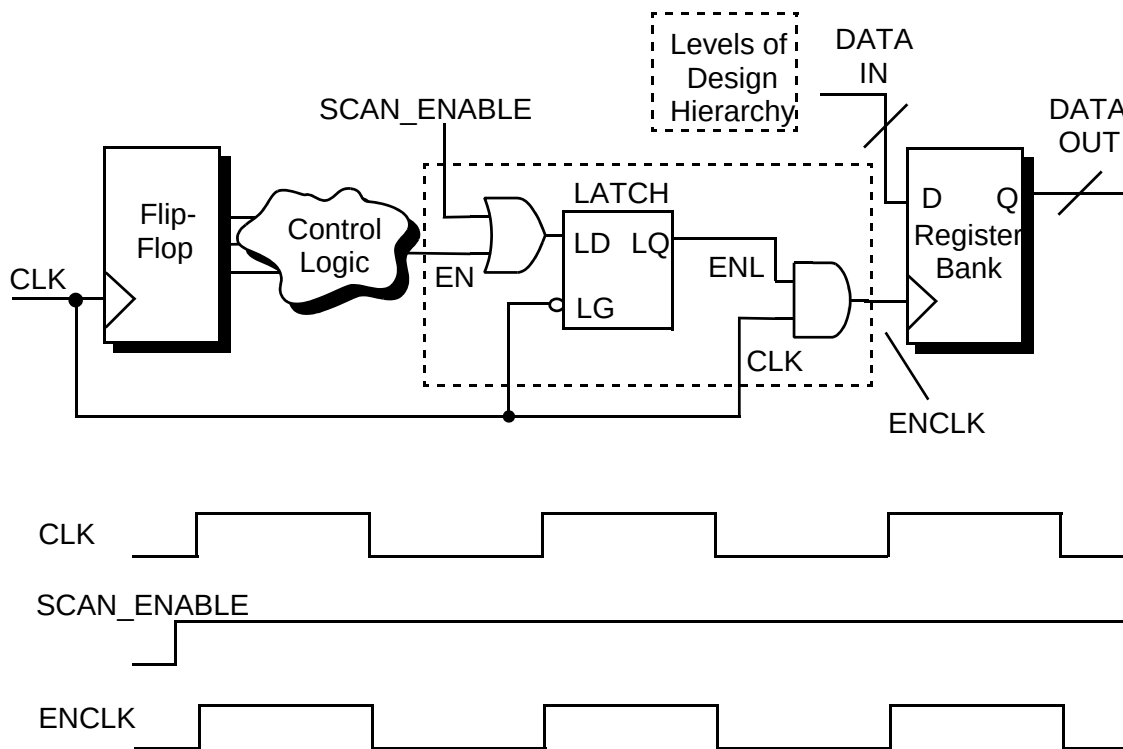
- Add a control point for testing
- Choose `test_mode` or `scan_enable`
- Add observability logic

Inserting a Control Point for Testability

A control point increases the testability of your design by restoring the clock signal to its ungated form during test. The control point is an OR gate that eliminates the function of the clock gate during test, which restores the controllability of the clock signal.

[Figure 7-11](#) shows a control point (OR gate) connected to the `scan_enable` port. The control point is before the latch in this example.

Figure 7-11 Control Point in Gated Clock Circuitry



When the `scan_enable` signal is high, the test signal overrides clock gating, thus making the `ENCLK` and `CLK` signals identical during shift mode. The test solution in [Figure 7-11](#) has the

advantage of achieving testability with the addition of only one OR gate. This configuration has fault coverage comparable to that of a design without clock gating.

The `set_clock_gating_style` command has two options to determine the location and type of the control point for test:

- `-control_point none | before | after`

The default is `none`. The `-control_point` option inserts your control point before or after the clock-gating latch. When using the latch-free clock-gating style, `before` and `after` are equivalent.

- `-control_signal test_mode | scan_enable`

The default is `scan_enable`. This option creates a `scan_enable` or `test_mode` test port and connects the port to the control-point OR gate. DFT Compiler interprets `test_mode` and `scan_enable` in a specific manner. The `-control_signal` option also applies to any observability logic inserted by the `-observation_point` option. You can use the `control_signal` option only if you have used the `-control_point` option.

When creating the control point, Power Compiler creates and names a new test port and assigns appropriate attributes to the port. [Table 7-5](#) shows variables that Power Compiler checks when naming the new port and when setting attributes on it.

Table 7-5 Test Port Naming and Attribute Assignment

Setting of <code>-control_signal</code>	Variable that determines test port name	Attributes on test port are the same as those set by
<code>scan_enable</code>	<code>test_scan_enable_port_naming_style</code>	<code>set_dft_signal -type ScanEnable</code>
<code>test_mode</code>	<code>test_mode_port_naming_style</code>	<code>set_attribute test_port_clock_gating</code> <code>set_dft_signal -type TestMode</code>

To connect the test port of the clock-gating design to the test port of your design, use the `insert_dft` command. For more information, see [“Connecting the Test Ports Throughout the Hierarchy” on page 7-42](#).

Latch-based clock gating requires that the enable signal always arrive after the trailing edge (rising edge for falling-edge signal) of the clock. If you insert the control point before the latch, it is impossible for the control point to violate this requirement. However, your test tool might not support positioning the control point before the clock-gating latch. In such cases, use `-control_point after` to insert the control point after the clock-gating latch.

Note:

If you insert the control point after the latch, the `scan_enable` signal or `test_mode` signal must transition after the trailing edge (rising edge for falling-edge signal) of the clock signal during test at the foundry; otherwise glitches in their resulting signal corrupts the clock output.

Scan Enable Versus Test Mode

Scan enable and test mode differ in the following way:

- Scan enable is active only during the scan mode.
- Test mode is active during the entire test (scan mode and parallel mode).

Scan enable typically provides higher fault coverage than test mode. Fault coverage with scan enable is comparable to a circuit without clock gating. However, there can be situations in which you must use test mode. For example, you might need to use test mode if you place the control point before the latch and your test tool does not support this position of the control point with scan enable.

Improving Observability With Test Mode

When using test mode, the EN signal and other signals in the control logic are untestable. If your test methodology requires that you use `test_mode`, you might need to increase your fault coverage. You can increase fault coverage with test mode by adding observability logic during clock gating.

Note:

When using the `-control_signal scan_enable` option, increasing observability with observability logic is not necessary.

The `set_clock_gating_style` command has two options for increasing observability when using the `-control_signal test_mode` option:

- `-observation_point true | false`

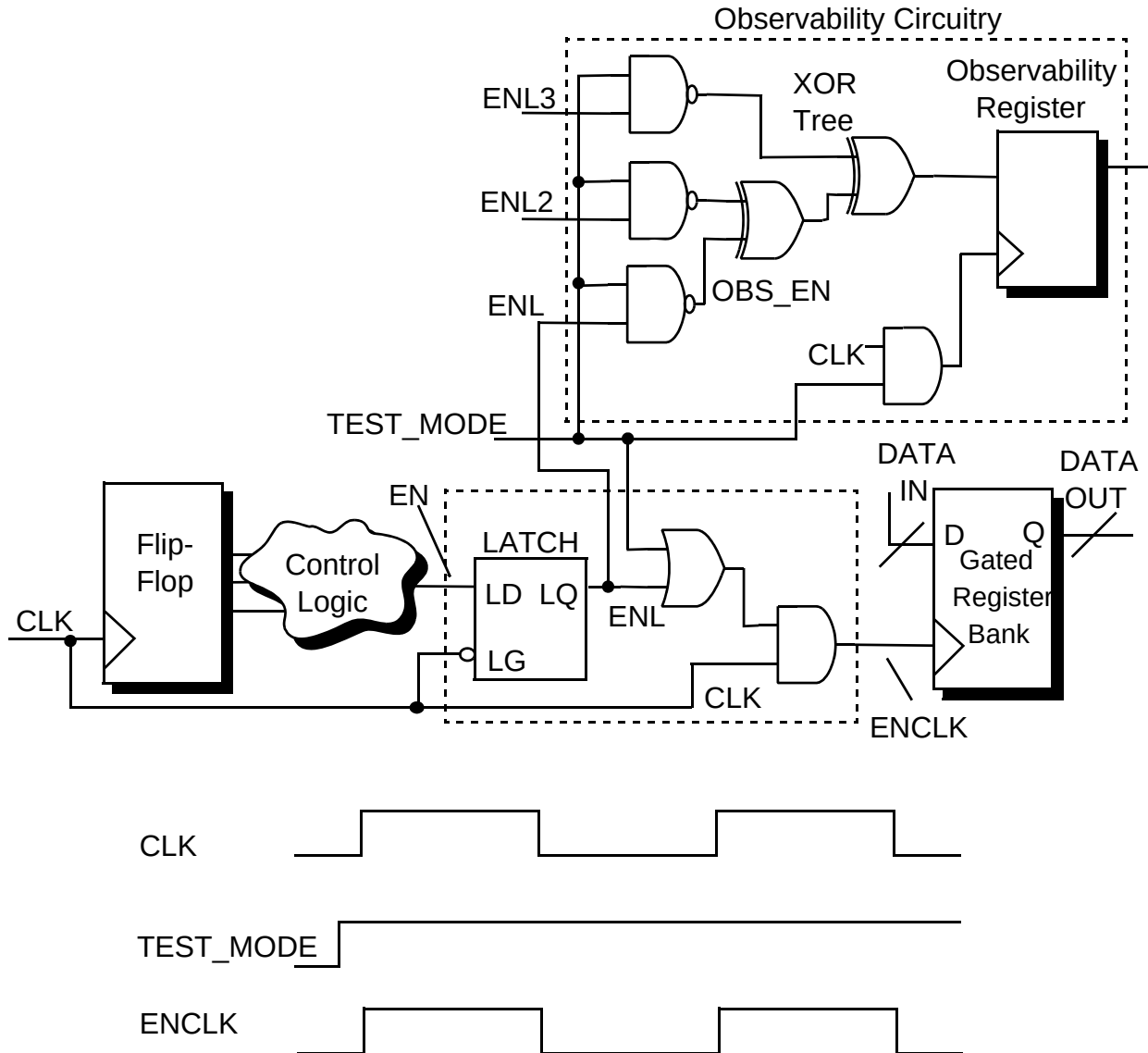
The default is `false`. When you set this option to `true`, clock gating adds a cell that contains at least one observability register and an appropriate number of XOR trees (if there is only one signal to be observed, an XOR tree is unnecessary). The scan chain includes the observability register, but the observability register's output is not functionally connected to the circuit.

- `-observation_logic_depth depth_value`

The default is 5. The value of this option determines the depth of logic of the XOR tree that `-observation_point` option builds during clock gating. If this value is set to 0, each ENL signal is latched separately and no XOR tree is built. The XOR tree reduces the number of observability registers needed to capture the test signature.

Figure 7-12 shows a gated clock, including an observability register and an XOR tree.

Figure 7-12 Gated Clock With High Observability



During test, observability circuitry allows observation of the ENL signal. During normal operation of the circuit, the XOR tree does not consume power, because the NAND gate blocks all ENL signal transitions. This test solution has high testability and is power-efficient, because the XOR tree consumes power only during test and the clock of the observability register is gated.

To connect the test port of the clock-gating design to the test port of your design, see [“Connecting the Test Ports Throughout the Hierarchy”](#) on page 7-42.

Choosing a Depth for Observability Logic

Use the `-observation_logic_depth` option of the `set_clock_gating_style` command to set the logic depth of the XOR tree in the observability cell. The default for this option is 5.

Power Compiler builds one observability cell for each clock-gated design. Each gated register in the design provides a gated enable signal (`OBS_EN` in [Figure 7-12 on page 7-41](#)) as input to the XOR tree in the observability cell.

If you set the logic depth of your XOR tree too small, clock gating creates more XOR trees (and associated registers) to provide enough XOR inputs to accommodate signals from all the gated registers. Each additional XOR tree adds some overhead for area and power.

If you set the logic depth of your XOR tree too high, clock gating can create one XOR tree with plenty of inputs. However, too large a tree can cause the delay in the observability circuitry to become critical.

Use the following guidelines in choosing or changing the logic depth of your XOR tree. Choose a value that is

- High enough to cause the construction of as few XOR trees as possible
- Low enough to keep the delay in the observability circuitry from becoming critical

Connecting the Test Ports Throughout the Hierarchy

You use the `insert_dft` command to connect the test ports through various level of the design hierarchy.

If you have used the clock-gating feature of Power Compiler with the testability options, you must connect the test ports using the `insert_dft` command. After you have compiled all the lower level hierarchies of the design, use the command on the top level of the design.

There are two types of test ports: the `test_mode` port and the `scan_enable` port. A port can be recognized as a test port if it is designated as a `scan_enable` or a `test_mode` port using the `set_dft_signal` command. Alternatively, a port can be designated as a test port by setting the `test_port_clock_gating` attribute on it.

A `scan_enable (test_mode)` port is only connected to other `scan_enable (test_mode)` ports in the design hierarchy. If a `scan_enable (test_mode)` port exists at a particular level of the hierarchy, it is connected to `scan_enable (test_mode)` ports at all higher levels of the hierarchy. If a `scan_enable (test_mode)` port does not exist at a higher level of hierarchy, the `scan_enable (test_mode)` port is created.

The `insert_dft` command connects the test ports on all levels of the design hierarchy to the `test_mode` or `scan_enable` pins of the OR gate in the clock gating logic and the XOR gates in the clock-gating observability logic. If the design does not have a test port at any level of hierarchy, a test port is created. If a test port exists, it is used.

Using the insert_dft Command

You use the `insert_dft` command to connect the top-level test ports to the test pins of the clock-gating cells through the design hierarchy. A test port is created if the design does not have a test port at any level of the hierarchy. To identify the test ports, the tool uses the options you specified using the `set_dft_signal` command. The following example shows the usage of the `insert_dft` command to connect to the clock-gating cells. When you specify the value `clock_gating` to the `-usage` option of the `set_dft_signal` command, during the execution of the `insert_dft` command, the tool connects the specified signal to the test pin of the clock-gating cells.

```
dc_shell> read_ddc design.ddc
dc_shell> set_clock_gating_style -control_signal scan_enable \
    -control_point before
dc_shell> compile_ultra -scan -gate_clock
dc_shell> set_dft_signal -type ScanEnable -port test_se_1
dc_shell> set_dft_signal -type ScanEnable -port test_se_2 \
    -usage clock_gating
dc_shell> create_test_protocol
dc_shell> dft_drc -verbose
dc_shell> preview_dft
dc_shell> insert_dft
```

For more information, see *DFT Compiler User Guide*.

Using Instance-Specific Clock-Gating Style

Power Compiler supports setting and removing clock-gating styles on specific design instances and on power domains. You can also enable and disable clock gating by overriding the specified styles. These instance-specific clock-gating styles are honored only by the `compile_ultra -gate_clock` command, as described in the following sections:

- [Specifying Clock-Gating Style on Design Objects](#)
- [Instance-Specific Clock-Gating Style Example](#)
- [Removing the Instance-Specific Clock-Gating Style on Design Objects](#)

Specifying Clock-Gating Style on Design Objects

The clock-gating style specified using the `set_clock_gating_style` command are applied to the entire design by default. To restrict the clock-gating style to specific objects of the design, follow these steps:

1. Set the `power_cg_iscgs_enable` variable to `true`. The default is `false`.
2. Use the `-instances` or the `-power_domains` option of the `set_clock_gating_style` command to restrict the clock-gating styles to be applied to the specified instances or power domains, respectively.

The clock-gating cells are inserted, based on the clock-gating style that you specified.

When you set the `power_cg_iscgs_enable` variable set to `true`, and a specific instance does not have a specified clock-gating style, the tool chooses a clock-gating style in the following decreasing order of priority:

- The style specified on the power domain containing the instance
- The style of the hierarchical cell containing the instance
- The style of the higher level hierarchical cell contains the instance
- When you do not specify the clock-gating style, Power Compiler derives a default clock-gating style based on the specified libraries. For more information, see [“Default Clock-Gating Style” on page 7-24](#).

Note:

If you set the `power_cg_iscgs_enable` variable to `true`, and do not use the `-instances` or the `-power_domains` option, the clock-gating style is applied only to the current design.

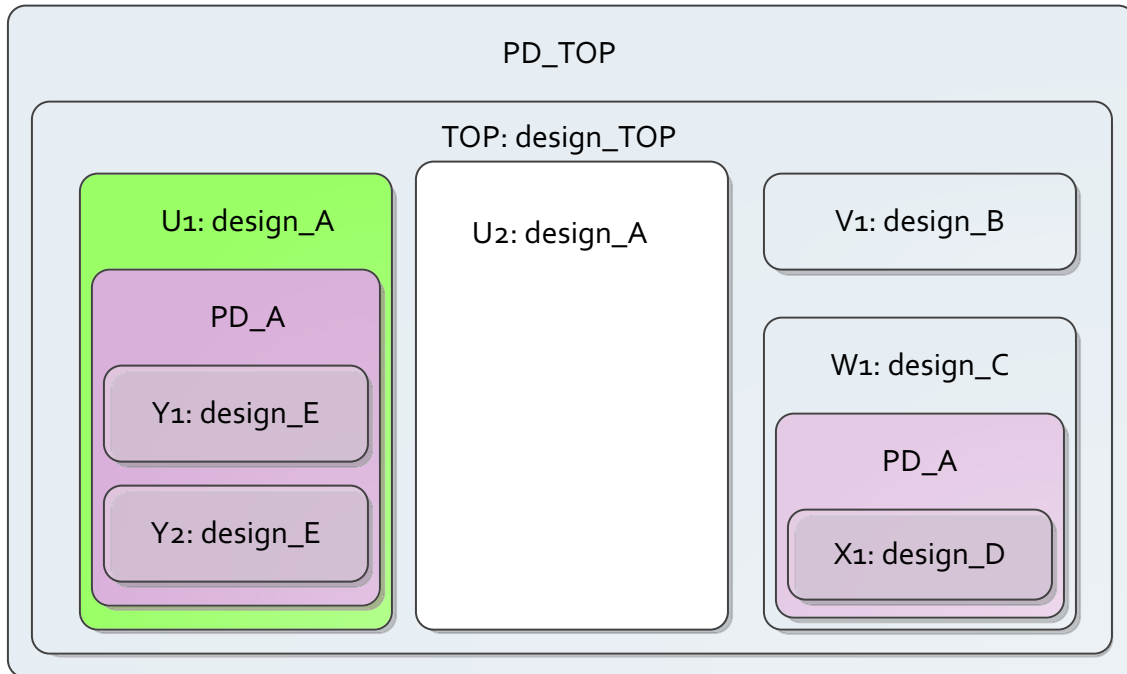
If you use the `-instances` or the `-power_domains` option of the `set_clock_gating_style` command without setting the `power_cg_iscgs_enable` variable to `true`, Power Compiler issues the PWR-815 error message.

Instance-Specific Clock-Gating Style Example

For the design example in [Figure 7-13](#), the `set_clock_gating_style` command is specified as follows:

```
# Specify the clock gating Style
dc_shell> set_clock_gating_style -designs {design_A}
dc_shell> set_clock_gating_style -instances {Y2}
dc_shell> set_clock_gating_style -instances {U1}
dc_shell> set_clock_gating_style -power_domains {PD_A}
```

Figure 7-13 Instance Specific Clock-Gating Style Example



The priority rules defined is applied from the bottom of the hierarchy to the top. Also, Power Compiler considers power domains to be more specific than design instances. In the design example of [Figure 7-13](#),

- The clock-gating style specified for Y2 instance is applied to the Y2 instance.
If a clock-gating style is defined for a hierarchical cell inside the instance Y2, the clock-gating style of the hierarchical cell is applied to Y2. This is because, the precedence rule is applied from the bottom of the hierarchy to the top.
The clock-gating style specified for Y2 instance has higher precedence than clock-gating style defined for PD_A power domain.
- The clock-gating style specified for U1 instance is applied to U1 instance, design design_A design, and PD_A power domain.
- The clock-gating style specified for the TOP design is applied to W1 and V1 instances.

Removing the Instance-Specific Clock-Gating Style on Design Objects

Use the `remove_clock_gating_style` command to remove the instance-specific clock-gating style that you specified on the design objects. However, this command can be used only when you set the `power_cg_iscgs_enable` variable to `true`.

Modifying the Clock-Gating Structure

While performing RTL clock gating, you can specify the `set_clock_gating_style -max_fanout` command to limit the number of registers that are gated by a single clock-gating element. The results can be multiple clock-gating elements that have the same enable signal and, logically, the same gated-clock signal. All clock-gating cells with the same enable signal belong to the same clock-gating group. All registers gated by a single clock-gating element belong to the same clock-gating subgroup.

The gated registers inserted by the `compile_ultra -gate_clock` command are partitioned into subgroups. These partitions are not based on timing or placement constraints. So the placement tool tries to place the clock-gated registers close to the clock-gating cell, but this might not happen because of other design constraints. The result is a suboptimal partition of gated registers into subgroups.

You can correct this problem by moving clock-gated registers between the clock-gating cells belonging to the same clock-gating group. Because these clock-gating cells are logically equivalent, the rewired circuit is functionally valid.

To rewire or remove clock gating in your design, use the `rewire_clock_gating` or `remove_clock_gating` command.

Changing a Clock-Gated Register to Another Clock-Gating Cell

To selectively rewire a clock-gated register from one clock-gating cell to another logically equivalent clock-gating cell, use the `rewire_clock_gating` command.

However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not perform rewiring of such clock-gating cells.

You can use the `-undo` option to remove any rewiring you specified with the `rewire_clock_gating` command. Based on the options specified, the `-undo` option deletes the directives specified by the previously specified `rewire_clock_gating` command. Use the `-undo` option before you use the `compile -incremental` command. The `compile` command modifies the netlist to rewire the gated registers.

Because rewiring the gated registers alters the clock-gating cell that gates the registers, any path-based timing exception that goes through the old clock-gating cell to a gated register is no longer relevant and is lost.

Removing Clock-Gating Cells From the Design

Power Compiler performs clock gating at the RTL level during the compilation process when you use the `compile_ultra -gate_clock` command. The `remove_clock_gating` command lets you selectively remove the clock gates without having to start at RTL again. The subsequent `compile_ultra` command removes the selected clock-gating cells. As a result you have the ability to use aggressive clock-gating strategies initially and selectively remove clock-gating cells, if needed.

This command removes redundant clock-gating cells that are no longer connected to any clock-gating cells. Any associated test observation logic is also optimized. However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not remove such cells.

All the registers that are not driven by the clock-gated signals are remapped to new sequential cells. This might result in new pin names for the registers. If there are pin-based timing exceptions set on the original register, these exceptions might not transfer properly during the transformation, if the new and the original pin names do not match. Pin-based timing exceptions are specified by using the `set_max_delay`, `set_min_delay`, `set_multicycle_path`, and `set_false_path` commands.

The `remove_clock_gating` command displays a warning if there are pin-based timing exceptions on the register to be ungated. Cell based timing exceptions are not affected because the ungated registers retain their name. It is advisable to use the cell-based timing exceptions with clock-gating registers.

For information, see the Design Compiler documentation.

Rewiring Clock Gating After Retiming

Power Compiler supports the `-balance_fanout` option to the `rewire_clock_gating` command.

This command is used to rebalance the fanout of the clock gates within the design after modifications have been made during retiming. During optimization, Power Compiler automatically balances the register banks based on the minimum and maximum fanout requirements. However, when you run the `compile -ungroup` or `optimize_registers` commands that perform retiming, unconnected registers are removed to improve timing. For clock tree synthesis, ensure that the clock gates have equivalent fanout loads by using the `-balance_fanout` option.

Use the `rewire_clock_gating -balance_fanout` command either after retiming or after compilation to restore a balanced fanout. When you use this command, Power Compiler compares the changed fanout of each equivalent clock-gating cell. The registers are moved around so that each equivalent clock-gating cell now has a balanced set of registers and

honors the `-max_fanout` option that you specified originally. Any register banks not meeting the `minimum_bitwidth` requirement are ungated. However, if a `dont_touch` attribute is set on a clock-gating cell or any of its parent in the hierarchy, the tool does not perform fanout balancing on such cells.

Note:

The command is not intended for use after the `balance_registers` command.

Integrated Clock-Gating Cells

An integrated clock-gating cell integrates the various combinational and sequential elements of a clock gate into a single cell located in the logic library. An integrated clock-gating cell is a cell that you or your library developer creates to use especially for clock gating.

Consider using an integrated clock-gating cell if you are experiencing timing problems, such as clock skew, caused by the placement of clock-gating cells on your clock line.

Use Library Compiler to create an integrated cell for clock gating. For detailed information, see the Library Compiler documentation.

Library Compiler assigns a black box attribute to the complex sequential cells such as integrated clock-gating cells. Design Compiler does not use the integrated cells for the general logic synthesis. Power Compiler uses these integrated clock-gating cell for clock-gating. The selection of the clock-gating cell is determined either by the default or the values specified with the `set_clock_gating_style` command. Each integrated clock-gating cell in the library must contain the Library Compiler attribute called `clock_gating_integrated_cell`. This attribute can be set to either the string `generic` or to one of 26 strings that represent specific clock-gating types. The string `generic` causes Library Compiler to infer the `clock_gating_integrated_cell` attribute from the functionality of the clock-gating cell. Using one of the 26 standard strings specifies the functionality explicitly according to established conventions. For more details, see [Appendix B, "Integrated Clock-Gating Cell Example."](#)

Integrated Clock-Gating Cell Attributes

The `clock_gating_integrated_cell` attribute should be set to one of 26 function-specific strings, such as `latch_posedge_postcontrol`. Each string is a concatenation of up to four strings that describe the cell's functionality. The library developer specifies the attribute when the integrated cell is created. When you set the `clock_gating_integrated_cell` attribute to `generic`, Power Compiler infers the value from Library Compiler.

For more information, see the *Library Compiler User Guide*.

The `clock_gating_integrated_cell` attribute can have any one of 26 different values. [Table 7-6](#) contains a short list of example values and their meanings.

Table 7-6 Examples of Values for Integrated Clock-Gating Cell

Value of <code>clock_gating_integrated_cell</code>	Integrated cell must contain
<code>latch_negedge</code>	Latch-based gating logic Logic appropriate for gating falling-edge-triggered registers
<code>latch_posedge_postcontrol</code>	Latch-based gating logic Logic appropriate for gating rising-edge-triggered registers Test control logic located after the latch
<code>latch_negedge_precontrol</code>	Latch-based gating logic Logic appropriate for gating falling-edge-triggered registers Test control logic located before the latch
<code>none_posedge_control_obs</code>	Latch-free gating logic Logic appropriate for gating rising-edge-triggered registers Test control logic (no latch) Observability port

For more examples, see [Appendix B, “Integrated Clock-Gating Cell Example.”](#)

The `set_clock_gating_style` command determines the integrated cell that Power Compiler uses for clock gating. Power Compiler searches the library for the integrated cell having the attribute value corresponding to the options you specify with the `set_clock_gating_style` command.

When you set the clock-gating style as

```
set_clock_gating_style
-sequential_cell latch
-positive_edge_logic {integrated}
-control_point before
-control_signal test_mode]
-observation_point true
```

The `-sequential_cell latch` and `-control_point before` result in a latch-based style, and the tool searches for an integrated clock-gating cell with `control` as the third string parameter of the `clock_gating_integrated_cell` attribute.

The tool selects a latch-based posedge integrated clock-gating cell with control point before, with observability pin because you specified `-positive_edge_logic {integrated}`.

If more than one integrated cell has the correct attribute value, Power Compiler chooses the first integrated cell that it finds in the target library. If you have a preference, specify the integrated cell by name.

Power Compiler does not check the function of the integrated cell to ensure that it complies with the value of the `clock_gating_integrated_cell` attribute. The correct functionality is checked by Library Compiler when the integrated cell is initially created. Power Compiler searches for an integrated clock-gating cell that contains the specified attribute value.

Pin Attributes

Power Compiler requires certain Library Compiler attributes on the pins of your integrated clock-gating cell. [Table 7-7](#) lists the required pin attributes for pin names that pertain to clock gating. Some pins, such as the pins for test and observability are optional; however, if a pin is present, it must have the corresponding attribute listed in [Table 7-7](#).

Table 7-7 Pin Attributes for Integrated Clock-Gating Cells

Integrated cell pin name	Input or output	Required Library Compiler attribute
clock	Input	clock_gate_clock_pin
enable	Input	clock_gate_enable_pin
test_mode or scan_enable	Input	clock_gate_test_pin
enable_clock	Output	clock_gate_out_pin
observability	Output	clock_gate_obs_pin

Other tools used in your synthesis and verification flow might require additional pin attributes that are not specific to clock gating and are not listed in [Table 7-7](#).

For more information about Library Compiler attributes and library syntax, see the Library Compiler documentation.

Timing Considerations

Clock gating requires certain timing arcs on your integrated clock-gating cell.

For latch-based clock gating,

- Define setup and hold arcs on the enable pin with respect to the clock pin.
For the latch-based gating style, these arcs are defined with respect to the controlling edge of the clock that is driving the latch.
- Define combinational arcs from the clock and enable inputs to the output.

For latch-free clock gating,

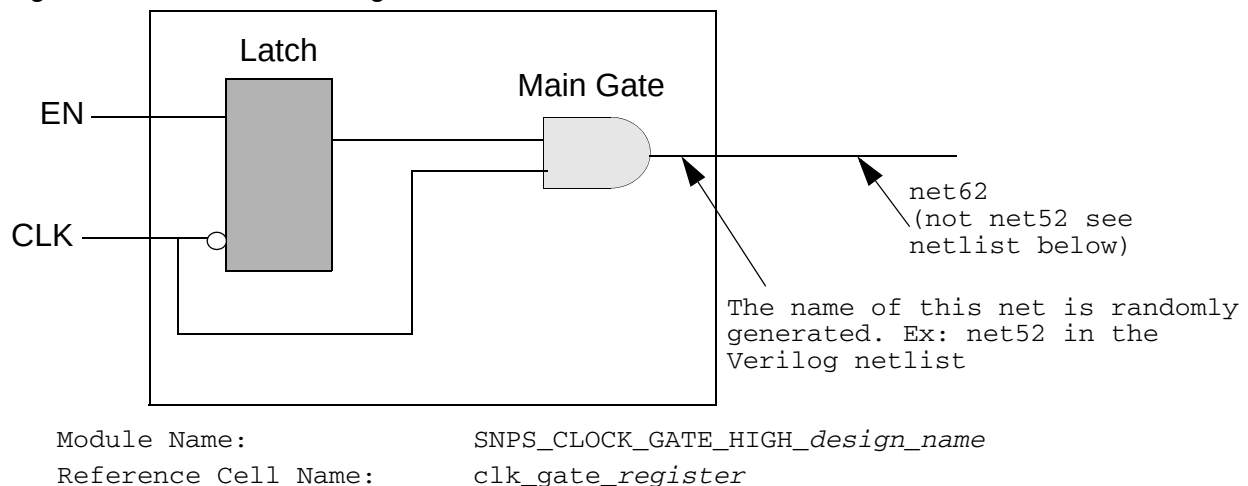
- Define no-change arcs on the enable pin with respect to the clock pin.
For the integrated latch-free gating style, these arcs must be no-change arcs, because they are defined with respect to different clock edges.
- Define combinational arcs from the clock and enable inputs to the output.

For more detailed information about timing your integrated cell, see the Library Compiler documentation.

Clock-Gating Naming Conventions

Clock-gating creates subdesigns containing clock-gating logic. Default naming conventions are shown in [Figure 7-14](#).

Figure 7-14 Default Naming Conventions



The Verilog netlist looks as follows:

```
module SNPS_CLOCK_GATE_HIGH_ff_03 ( CLK, EN, ENCLK );
    input CLK, EN;
    wire net50, net52, net53, net56;
    assign net50 = CLK;
    assign net50 = CLK;
    assign ENCLK = net52;
    assign net53 = EN;

    L_CSLDP1NQW latch ( .D(net53), .ENN(net50),
.Q(net56) );
    L_CSAN2 main_gate ( .A(net56), .B(net50), .Z(net52) );
endmodule
module ff_03 ( q, d, clk, e, clr );
    output [2:0] q;
    output [2:0] q;
    input [2:0] d;
    input clk, e, clr;
    wire N0, net62;

    L_CSFD2QP \q_reg[2] ( .D(d[2]), .CP(net62), .RN(clr),
.Q(q[2]) );
    L_CSFD2QP \q_reg[1] ( .D(d[1]), .CP(net62), .RN(clr),
.Q(q[1]) );
    L_CSFD2QP \q_reg[0] ( .D(d[0]), .CP(net62), .RN(clr),
.Q(q[0]) );
    SNPS_CLOCK_GATE_HIGH_ff_03 clk_gate_q_reg ( .CLK(clk),
.EN(N0),
.ENCLK(net62) );
    L_CSIV1 U5 ( .A(e), .Z(N0) );
endmodule
```

The module_name(SNPS_CLOCK_GATE_..), reference cell_name(clk_gate..) and the gated_clock enable net name(net62) could be changed according to your preferences.

Set the power_cg_module_naming_style, power_cg_cell_naming_style, and power_cg_gated_clock_net_naming_style variables before running the insert_clock_gating command.

Use the variables either in .synopsys_setup.dc file or before clock-gate insertion. The details of the implementation are as follows:

```
Usage: set power_cg_module_naming_style
"prefix_%e_%l_midfix_%p_%t_%d_suffix"
    where,
    prefix/midfix/suffix are just examples of any constant
strings that can
be specified.
    %e - edge type (HIGH/LOW)
    %l - library name of integrated clock gating cell library
or concatenated target_library names
    %p - immediate parent module name
```

%t - top module (current design) name
 %d - index added if there is a name clash

Usage: set power_cg_cell_naming_style
 "prefix_%c_%n_midfix_%r_%R_%d_suffix"
 where,
 %c - clock
 %n - immediate enable signal name
 %r - first gated reg bank name
 %R - all gated reg banks sorted alphabetically
 %d - index for splitting or name clash resolution

Usage: set power_cg_gated_clock_net_naming_style
 "prefix_%c_%e_%g_%d_suffix"
 %c - original clock
 %e - immediate enable signal name
 %g - clock gate (instance) name
 %d - index for splitting or name clash resolution

Note:

If %d is not specified, Power Compiler assumes a %d at the end.

Example Script for Naming Style

```
set power_cg_module_naming_style Synopsys_%e_mid_%t
set power_cg_cell_naming_style cg_%c_%n_mid_%R
set power_cg_gated_clock_net_naming_style gclk_%c_%n

define_design_lib WORK -path ./work_writable
set target_library  cstarlib_lvt.db
set link_library    { cstarlib_lvt.db }

set_clock_gating_style -sequential_cell latch -max_fanout 3 \
  -minimum_bitwidth 1
analyze -format verilog -library WORK ff_03.v
elaborate ff_03
insert_clock_gating
uniquify
create_clock -name "clk" -period 5 \
  -waveform {"0" "2.5" } { "clk" }
compile_ultra
current_design ff_03
write -format verilog -output 3.ff_03.vg -hierarchy
```

Example Script of Output Netlist

```
module Synopsys_HIGH_mid_ff_03_0 ( CLK, EN, ENCLK );
  input CLK;
  input EN;
```

```

output ENCLK;
wire net15, net12, net11, net9;
assign net12 = EN;
assign ENCLK = net11;
assign net9 = CLK;

L_CSAN2 main_gate ( .A(net15), .B(net9), .Z(net11) );
L_CSALDP1NQW latch ( .D(net12), .ENN(net9), .Q(net15) );
endmodule

module ff_03 ( q, d, clk, e, clr );
output [2:0] q;
input [2:0] d;
input clk;
input e;
input clr;
wire N1, gclk_clk_N1_0;

Synopsys_HIGH_mid_ff_03_0 cg_clk_N1_mid_q_reg_0 (
.CLK(clk), .EN(N1),
.ENCLK(gclk_clk_N1_0) );
L_CSFD2QP \q_reg[2] ( .D(d[2]), .CP(gclk_clk_N1_0),
.RN(clr), .Q(q[2])
);
L_CSFD2QP \q_reg[1] ( .D(d[1]), .CP(gclk_clk_N1_0),
.RN(clr), .Q(q[1])
);
L_CSFD2QP \q_reg[0] ( .D(d[0]), .CP(gclk_clk_N1_0),
.RN(clr), .Q(q[0])
);
L_CSIV1 U3 ( .A(e), .Z(N1) );
endmodule

```

Keeping Clock-Gating Information in a Structural Netlist

Power Compiler applies several clock-gating attributes to the design and to the clock-gating cells and gated registers in the design. Commands such as `report_clock_gating`, `rewire_clock_gating`, `remove_clock_gating` and several placement optimization algorithms depend on these attributes for proper operation.

The `power_cg_flatten` variable specifies whether to flatten the clock-gating cells when you use commands that perform ungrouping, such as `ungroup`, `compile -ungroup_all`, or `balance_registers`. By default, the variable is set to `false` and the clock-gating cells are not flattened. This is recommended for most situations because ungrouping the discrete clock gates could cause problems.

You can write a clock-gated structural netlist in ASCII format after synthesis. Reading back the structural netlist in ASCII format causes the clock-gating attributes to be lost, possibly preventing clock-gating and optimization from operating properly.

Power Compiler can automatically retrieve the clock-gating attributes and identify the clock-gating cells when you read the ASCII netlist. For more information, see [“Identifying and Preserving Clock-Gating Cells” on page 7-55](#).

Identifying and Preserving Clock-Gating Cells

The clock-gate identification feature helps the tool to recognize the clock-gating cells that it inserted in the netlist in the previous run or the clock-gating cells that you instantiated in the ASCII netlist.

Identification of Clock-Gating Cells

Power Compiler identifies clock-gating cells, including the hierarchical integrated clock-gating cells that exist in an ASCII netlist or the discrete hierarchical clock-gating cells inserted in the previous run of the tool.

For Power Compiler to identify the clock-gating cells inserted by the tool and annotate the related attributes, either set the `power_cg_auto_identify` variable to `true` or use the `identify_clock_gating` command without specifying any option.

The following example shows how to use the `power_cg_auto_identify` variable to identify and report the identified clock-gating cells.

```
set power_cg_auto_identify true
read_verilog my_design.v
current_design my_design_top
report_clock_gating
```

For more details, see the `power_cg_auto_identify` variable man page and the `identify_clock_gating` command man pages.

Explicit Identification of Clock-Gating Cells

When using the `identify_clock_gating` command without options cannot identify clock-gating cells, specify the `-gating_elements` option with the command to explicitly identify clock-gating cells. The tool sets the `pwr_cg_preservation_type` attribute on the specified cell.

To explicitly identify a clock-gating cell, the cell must have at the least two input pins and one output pin; one of the input pins must be a clock pin.

Explicit identification provides you the flexibility to identify the clock-gating cells that differ from the configuration expected by the Power Compiler tools for automatic identification. However the explicitly identified clock-gating cells have additional optimization restrictions.

When a cell could be automatically identified but was explicitly identified, the Power Compiler tool sets the `pwr_cg_preservation_type` attribute to `preserve` on the cell, and

you can remove the attribute. When a cell is identified explicitly and could not be automatically identified, the tool sets the `pwr_cg_preservation_type` attribute to `unmodifiable_read_only` and you cannot remove the attribute.

For more information on the `pwr_cg_preservation_type` attribute, see [“Preserving the Identified Clock-Gating Cells” on page 7-56](#).

The following example shows how to identify a specific clock-gating element using the `-gating_elements` option of the `identify_clock_gating` command.

```
dc_shell> read_verilog design.v
dc_shell> current_design top
dc_shell> link
# Defining the clock is not a prerequisite for clock-gate identification
# Identifies all the clock-gating cells inserted by the tool
dc_shell> identify_clock_gating

# Identifies the specified clock-gating cell
dc_shell> identify_clock_gating -gating_elements CG_1
dc_shell> report_clock_gating
```

For more details, see [“Usage Flow With the identify_clock_gating Command” on page 7-61](#).

Preserving the Identified Clock-Gating Cells

To preserve the identified clock-gating cells, use the `set_preserve_clock_gate` command. The command sets the `pwr_cg_preservation_type` attribute to one of the following values on the specified clock-gating cells:

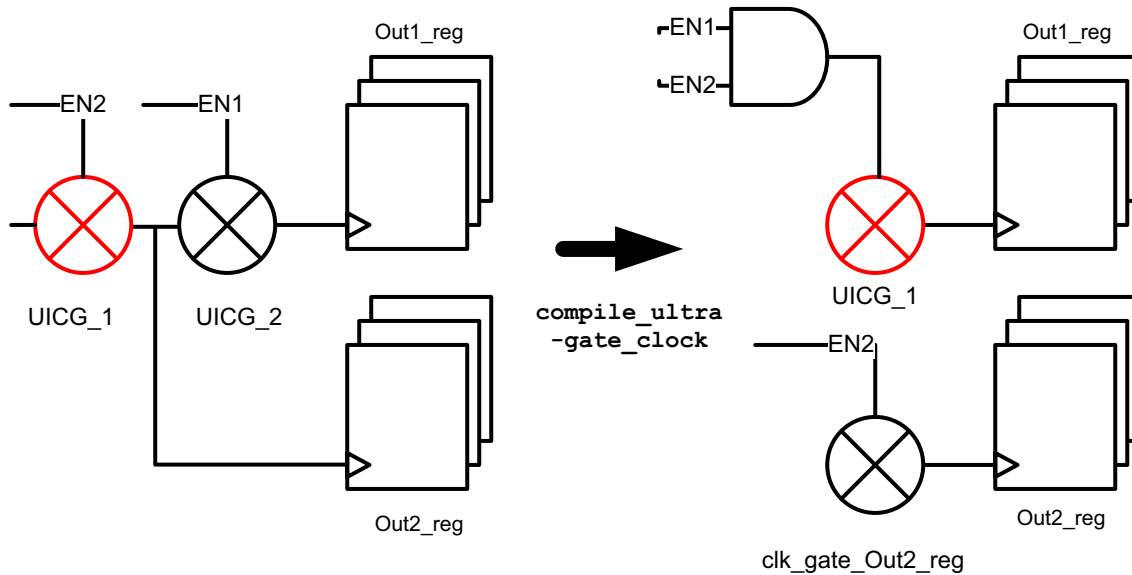
- `preserve`
- `dont_modify_fanout`
- `dont_modify_enable`
- `unmodifiable`

Note:

When you specify a non clock-gating cell with the `set_preserve_clock_gate` command, the Power Compiler tool ignores the command without any warning message.

The following example and [Figure 7-15](#) illustrate clock-gating optimization when the `pwr_cg_preservation_type` attribute is set to `preserve`:

```
set power_cg_reconfig_stages true
set_clock_gating_style -num_stages 1
identify_clock_gating
set_preserve_clock_gate [get_cell UICG_1]
```

Figure 7-15 Optimization When *pwr_cg_preservation_type* Is Set to *preserve*

The following table details the expected behavior between the option settings of the `set_preserve_clock_gate` command and the `pwr_cg_preservation_type` attribute values.

Table 7-8 Behavior of the *set_preserve_clock_gate* Command and the *pwr_cg_preservation_type* Attribute

set_preserve_clock_gate	pwr_cg_preservation_type	Behavior
No options given	preserve	Preserve the specified clock-gating cell; allow all clock-gating optimizations if the cell and its name are conserved after <code>compile_ultra-gate_clock</code> command.
-dont_modify_fanout	dont_modify_fanout	Preserve the specified clock-gating cell and prevent any clock-gating optimization in the direct fanout of the specified clock-gating cell.

Table 7-8 Behavior of the set_preserve_clock_gate Command and the pwr_cg_preservation_type Attribute (Continued)

set_preserve_clock_gate	pwr_cg_preservation_type	Behavior
-dont_modify_enable	dont_modify_enable	Preserve the specified clock-gating cell and prevent any clock-gating optimization in the clock-gating enable logic.
-dont_modify_fanout -dont_modify_enable	unmodifiable	Prevent any further clock-gating optimization on the clock-gating fanout and clock-gating enable logic of the specified clock-gating cell.

Note:

The -dont_modify_fanout and the -dont_modify_enable options restrict only clock-gating optimizations. Any other compile optimization is still allowed, such as remapping, buffering, boundary retiming, and so on.

When you use the set_preserve_clock_gate command for a clock-gating cell,

- If the -dont_modify_fanout option is not used, the tool optimizes the cell if it does not drive any load or does not meet the minimum bitwidth.
- The remove_clock_gating command does not remove these cells

During clock-gate merging, the Power Compiler tool preserves the cell that has the pwr_cg_preservation_type attribute set as shown in [Figure 7-25 on page 7-75](#).

To remove the pwr_cg_preservation_type attribute, use the remove_attribute command. However, you cannot remove it if the pwr_cg_preservation_type attribute is set to unmodifiable_read_only during explicit clock-gating identification.

For more details about the pwr_cg_preservation_type attribute, see the set_preserve_clock_gate command man page

Identified Clock-Gating Cells and dont_touch

When you use the set_dont_touch command on identified clock-gating cells, the cells are affected in the following ways:

- No rewiring of the clock gate
- No removal of the clock gate
- No merging or splitting of the clock gates

The `dont_touch` setting also affects the fanout of the clock-gating cells in the following ways:

- No further addition of clock-gating cells
- No fanout balancing of flip-flops that were gated
- No addition or removal of loads on the fanout of the clock-gating cell

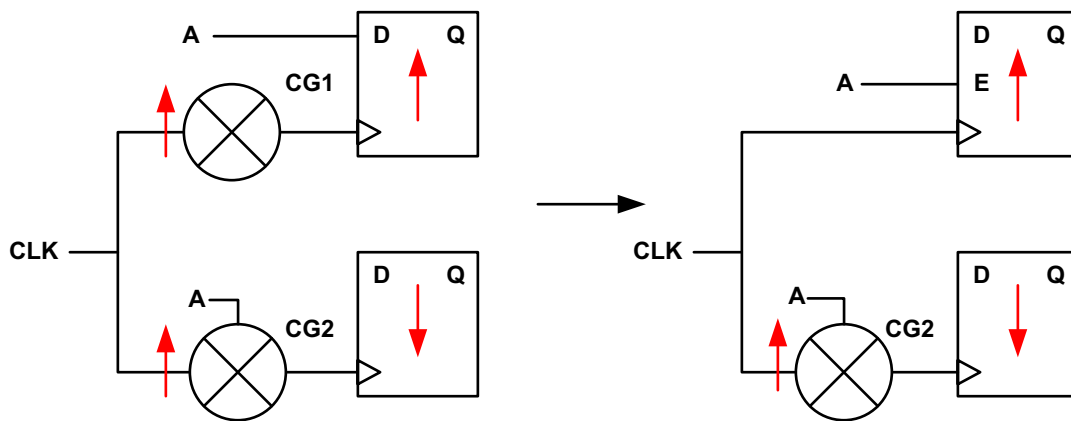
Handling Clock-Gating Edge Conflicts

Instantiated clock-gating cells that drive registers with a different activation edge are identified and optimized like any other clock-gating cell. The clock gating style for these cells are honored if they do not invert the clock signal. For example, suppose you have the following code:

```
set_clock_gating_style -minimum_bitwidth 3
identify_clock_gating
compile_ultra -gate_clock
```

With the example shown in [Figure 7-16](#), the tool removes the clock-gating cell named CG1 because its activation edge is the same as the register it gates. However, it does not remove the positive-edge clock-gating cell named CG2 because it gates a register with a negative-edge activation edge.

Figure 7-16 Optimization of Same-Edge Activation Clock-Gating Cell



Comparison of Clock-Gate Identification Methods

The advantages and disadvantages of the various methods of clock-gate identification are summarized in [Table 7-9](#).

Table 7-9 Identifying Clock-Gated Designs

Command Used	Advantages	Disadvantages
<code>write_script</code>	Clock-gating attributes are written using the <code>set_attribute</code> and <code>set_preserve_clock_gate</code> commands to save the current settings. This method uses familiar commands and procedure.	Netlist changes are not supported.
<code>identify_clock_gating</code>	Netlist changes performed outside of Design Compiler are supported.	You must run this command at the right place. Some attributes such as <code>max_fanout</code> might be lost unless the <code>set_clock_gating_style</code> command is used.

Usage Flow With the `write_script` Command

Follow these steps to retrieve the clock-gating information in the ASCII netlist using the `write_script` command.

1. Setup environment. Read in the RTL design. Insert the clock-gating logic.
2. Compile the design with the required constraints.
3. Run the `change_names` command to conform to the specified rules.
4. Write out the netlist.
5. Save current attributes and settings by using `write_script -hierarchy` command. Use the `-output` option of the command to write the output to a file. This command writes out all the attributes set by the `set_attribute` command.
6. Quit the Design Compiler session. Make sure you do not make any changes to the netlist before quitting.
7. Read in the design netlist.

8. Source the file written by the `write_script` command. This sets all the required attributes on the design, including the clock-gating cells, for proper execution throughout the flow.

If you do not need clock-gating information, you can use the `-no_cg` option of the `write_script` command. This results in a smaller script file.

To report the identified clock gates, use the `report_clock_gating` command.

The following example script shows the output file created by the `write_script` command.

```
#####
# Created by write_script -format dctl on August 26, 2016 11:16 am
#####
# Set the current_design #
current_design module4

set_local_link_library {CORELIB8DLL.db}
set_attribute -type int [current_design] power_cg_max_fanout
2048
set_attribute -type boolean [get_cells clk_gate_out1_reg] \
clock_gating_logic true
set_attribute -type boolean [get_cells clk_gate_out1_reg] \
hpower_inv_cg_cell false
set_attribute -type integer [get_cells {out1_reg[0]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[1]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[2]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[3]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[4]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells {out1_reg[5]}] \
power_cg_gating_group 0
set_attribute -type integer [get_cells clk_gate_out1_reg] \
power_cg_gating_group 0
set_size_only [get_cells latch] true
```

Usage Flow With the `identify_clock_gating` Command

This section describes the steps you follow to retrieve the clock-gating information using the `identify_clock_gating` command.

After you have saved the design that has the clock-gating information, follow these steps to retrieve the clock-gating information:

1. Read in the structural netlist that already has clock-gating cells inserted.
2. Set the `set_clock_gating_style` command. This ensures that the settings are the same as before saving the design. Otherwise, a few attributes such as `max_fanout` are not retained.
3. Use the `identify_clock_gating` command without any options to identify all clock-gating elements. This step traverses the design, searches appropriately for the clock-gating structure recognized by the power Compiler tool, and annotates the attributes needed for later operations.

Your design now contains all the clock-gating information. You can verify this using the `report_clock_gating` command.

Note:

Identify the clock-gating elements before optimizing the design so that the enable logic of the clock-gating elements can be optimized by the `compile_ultra -gate_clock` command.

Replacing Clock-Gating Cells

The Power Compiler tool detects clock-gating circuitry at the block or module level. At the module level, the clock-gating circuit can be either an instantiated or inferred logic. The tool replaces this logic with an integrated clock-gating cell or discrete cells according to the attributes that the `set_clock_gating_style` command specified. This cell replacement is performed using the `replace_clock_gates` command. This feature allows you to use the integrated clock-gating cell that is recognized by the `report_clock_gating`, `remove_clock_gating`, and `rewire_clock_gating` commands, for further operations.

Follow these steps to perform module-level replacement of clock-gating cells:

1. Set clock-gating directives and styles (optional).

The default settings of the `set_clock_gating_style` command is suitable for most designs. You can choose a value for the clock-gating conditions and a clock-gating style that is compatible with the clock-gating cell that is being replaced using the `set_clock_gating_style` command. Note that the `replace_clock_gates` command only operates on clock gate composition when using the style settings. It does not use the `-num_stages` option.

2. Read the RTL design.
3. Define the clock ports.

The clock port must be identified using the `create_clock` command before performing the replacement operation.

4. Replace manually-instantiated clock-gating cells.

To replace manually-inserted clock gates with tool-inserted clock gates, use the `replace_clock_gates` command. To perform the replacement hierarchically, use the `-global` option. If you have not specified a clock-gating style, the tool uses the default clock-gating style.

Note:

This command replaces only combinational logic. It does not replace observability logic.

5. Compile the design.

The replaced clock-gating logic is unmapped. Use the `compile_ultra` command to compile your design.

6. Report the gate elements registers.

Use the `report_clock_gating` command to get the list of cells as shown in the following example:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
dc_shell> report_power
```

In the following example, replacement is performed on a gating cell that is driving registers in a black box cell:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> set_replace_clock_gates -rising_edge_clock RAM/clk
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
```

In the following example, replacement is performed only on selected gating cells:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> set_replace_clock_gates -exclude_cells {SUB/C10}
dc_shell> replace_clock_gates
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating
```

[Example 7-1](#) shows a clock-gate replacement report.

Example 7-1 Clock-Gate Replacement Report

```

Current clock gating style....
Sequential cell: none
Minimum register bank size: 3
Minimum bank size for enhanced clock gating: 6
Maximum fanout: 2048
Setup time for clock gate: 1.300000
Hold time for clock gate: 0.000000
Clock gating circuitry (positive edge): or
Clock gating circuitry (negative edge): and
Note: inverter between clock gating circuitry
      and (negative edge) register clock pin.
Control point insertion: none
Control signal for control point: scan_enable
Observation point insertion: false
Observation logic depth: 5
Maximum number of stages: 5
1
replace_clock_gates -global
  Loading target library 'ssc_core_typ'
  Loading design 'regs'
Information: Performing clock-gating on design regs

```

Clock Gate Replacement Report

Clock Root	Cell Name	Include Exclude	Clock Fanin	Edge Type	Func.	Setup Cond.	Gate Repl.
clk	C7	-	1	fall	and	yes	yes

Summary:

	number	percentage
Replaced cells (total):	1	100
Cell not replaced because		
Cell was excluded:	0	0
Multiple clock inputs:	0	0
Mixed or unknown clock edge type:	0	0
No compatible clock gate available:	0	0
Setup condition violated:	0	0
Total:	1	100

Clock Gate Insertion Report

Gated Group	Flip-Flop Name	Include Exclude	Bits	Enable Cond.	Setup Cond.	Width Cond.	Clock Gated
cg0	GATED REGISTERS						
	q2_reg[3]	-	1	yes	yes	yes	yes
	q2_reg[2]	-	1				

cg1	q2_reg[1]	-	1	yes	yes	yes	yes
	q2_reg[0]	-	1				
	q3_reg[3]	-	1				
	UNGATED REGISTERS						
	si_reg	-	1	no	??	??	no
	ti_reg	-	1	no	??	??	no
	q4_reg[0]	-	1	no	??	??	no

Summary:

Flip-Flops

	Banks		Bit-Width	
	number	percentage	number	percentage
Clock gated (total):	3	30	12	54
Clock not gated because				
Bank was excluded:	0	0	0	0
Bank width too small:	0	0	0	0
Enable condition not met:	7	70	10	45
Setup condition violated:	0	0	0	0
Total:	10	100	22	100

Clock gates in design	number	percentage
Replaced clock gates:	1	16
Inserted clock gates:	3	50
Factored clock gates:	2	33
Total:	6	100

Multistage clock gating information

Number of multistage clock gates:	2
Average multistage fanout:	2.0
Number of gated cells:	16
Maximum number of clock gate stages:	3
Average number of clock gate stages:	2.2

Clock-Gate Optimization Performed During Compilation

To further increase the power saving of your design, Power Compiler uses certain techniques during compilation to reduce the number of clock-gating cells in the design. These techniques are described in detail in the following sections:

- [Hierarchical Clock Gating](#)
- [Enhanced Register-Based Clock Gating](#)
- [Multistage Clock Gating](#)
- [Clock Gate Merging](#)
- [Clock Gating Multibit Registers](#)
- [Placement-Aware Clock Gating in Design Compiler Graphical](#)

Hierarchical Clock Gating

Generally, clock-gating techniques in Power Compiler extract common enable conditions that are shared across the registers within the same block.

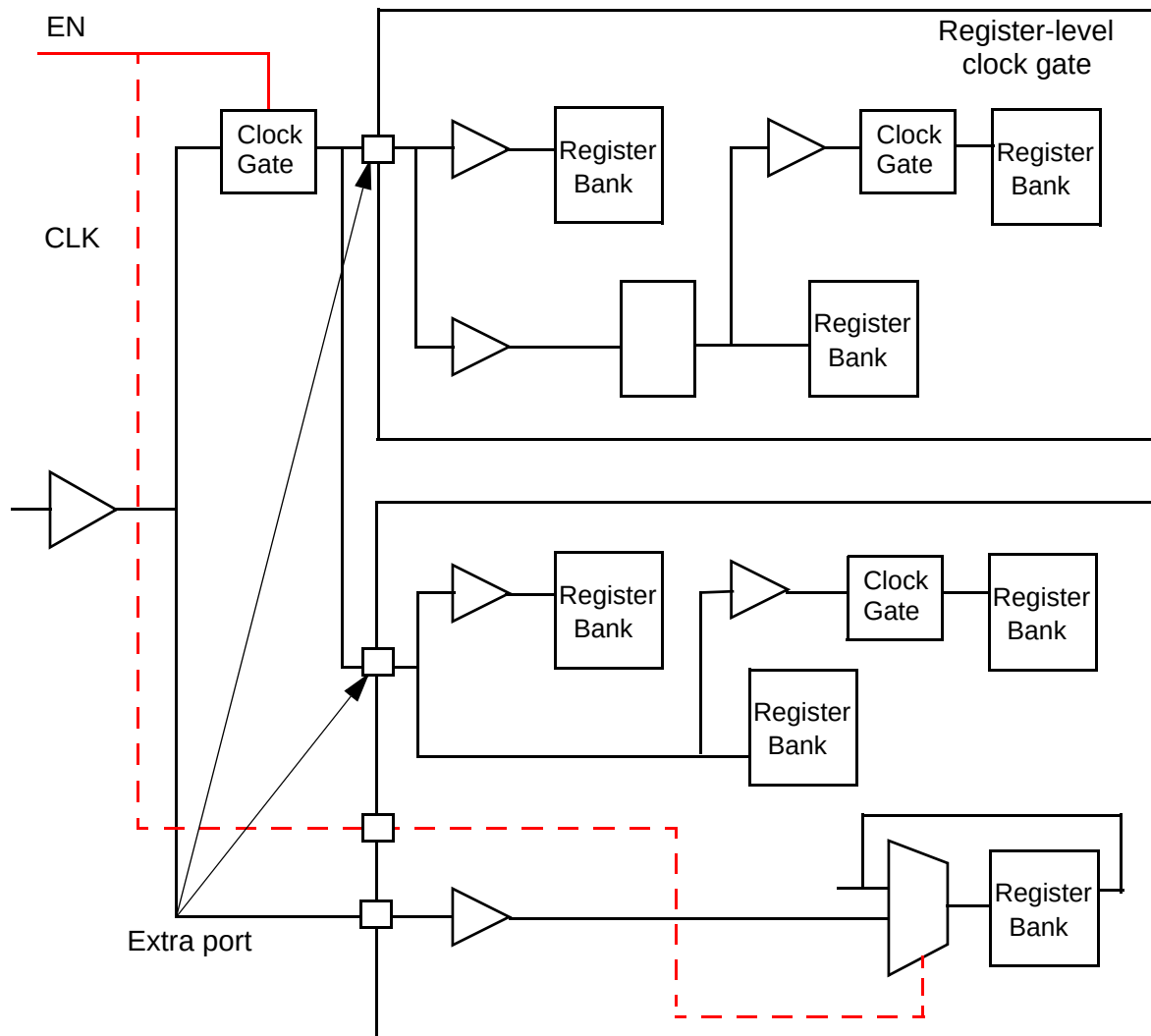
In hierarchical clock gating, during the clock-gate insertion, Power Compiler extracts the common enables shared across registers in different levels of hierarchy in the design. This technique looks for globally shared enables while inserting clock-gating cells. This increases the clock-gating opportunities and also reduces the number of clock gates inserted. This technique, combined with proper placement, improves the power savings.

Note:

During hierarchical clock gating, Power Compiler honors the boundary optimization settings. If you disable boundary optimization, Power Compiler does not perform hierarchical clock-gate insertion.

Power Compiler inserts hierarchical clock-gating cells at various levels of design hierarchy. As a result, additional ports are created for the clock-gated enable signal as shown in [Figure 7-17](#). These additional ports are added to the subdesigns. Formality verifies the designs successfully when the designs being compared have the same number of primary ports.

Figure 7-17 Ports Added During Hierarchical Clock Gating



Power Compiler can perform hierarchical clock gating on RTL netlists as well as gate-level netlists. To perform hierarchical clock gating by using the `compile_ultra -gate_clock` command, you must set the `compile_clock_gating_through_hierarchy` variable to `true` before compiling your design. If you use the `insert_clock_gating` command, you must use the `-global` option.

The following example shows hierarchical clock gating using the `compile_ultra` command:

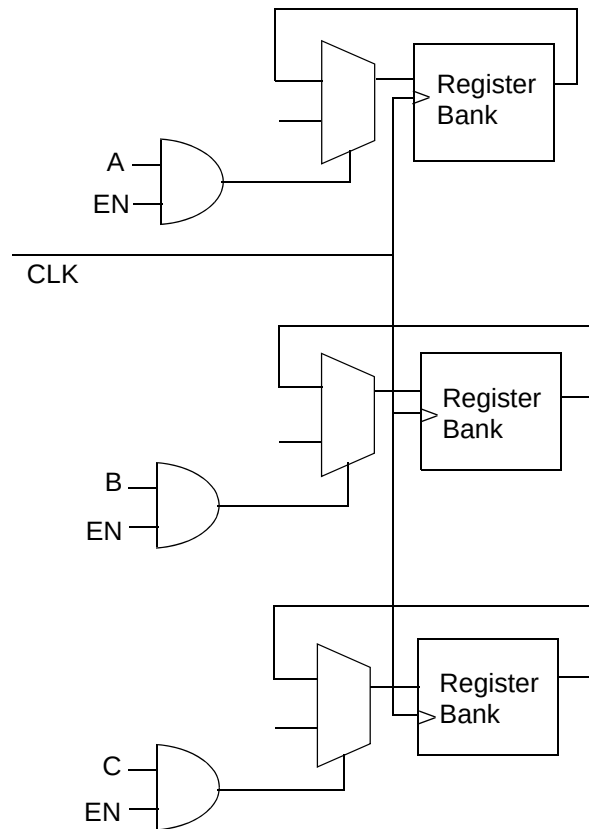
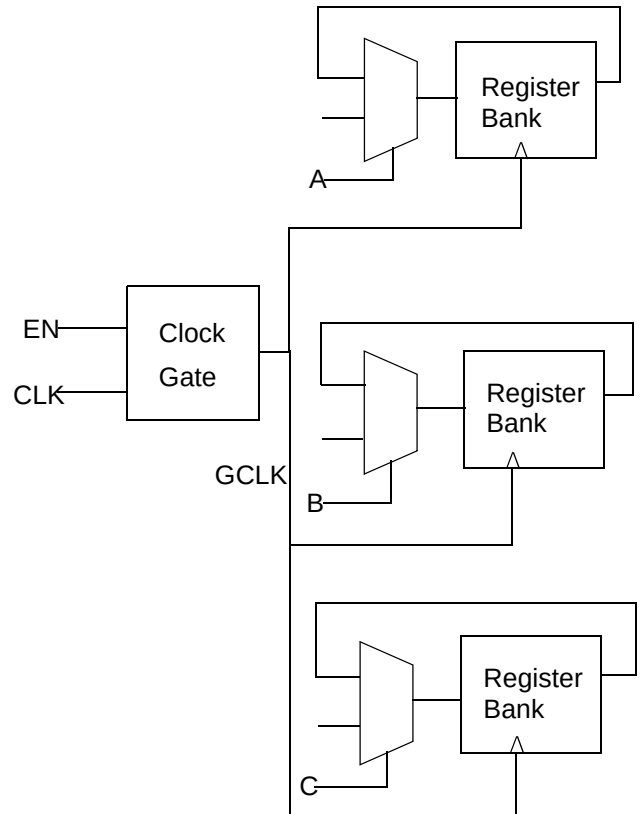
```
# Set your target library and link library
# set the clock-gating style (optional)
# Following command is optional. Use for global clock gating
dc_shell> set compile_clock_gating_through_hierarchy true
```

```
# Read your design
dc_shell> create_clock -name clk -period 10
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating -verbose -gating_elements -gated
dc_shell> report_power
```

Enhanced Register-Based Clock Gating

The regular register-based clock gating requires certain conditions in order for successful implementation. One of these conditions is the minimum bit-width of the register bank to be gated. If the minimum bit-width is less than 3, which is the default, there is no clock-gating opportunity. This width constraint ensures that the overhead of using the clock-gating cell does not overcome the power savings.

Power Compiler can factor out the common enable signal EN shared between three register banks and insert one clock-gating cell for these register banks, which would normally not be clock gated due to the width condition. The result is shown in [Figure 7-18](#).

Figure 7-18 Design With Common Enable Signal*Width Condition Violation (W=2):**No Clock Gating**Common Enable Factoring*

The default total minimum bit-width of registers for enhanced clock gating to be implemented is twice that of regular clock gating. Since the default value for regular register clock gating is 3, for the enhanced clock gating the register width should be at least $2 * 3$, which is 6.

Enhanced clock gating is the default behavior when you use the `insert_clock_gating` command. To disable enhanced clock gating, use the `-regular_only` option of the `insert_clock_gating` command.

In the following example, automated clock gating, with enhanced clock gating is implemented if the clock-gating conditions are met.

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> insert_clock_gating
dc_shell> report_clock_gating
```

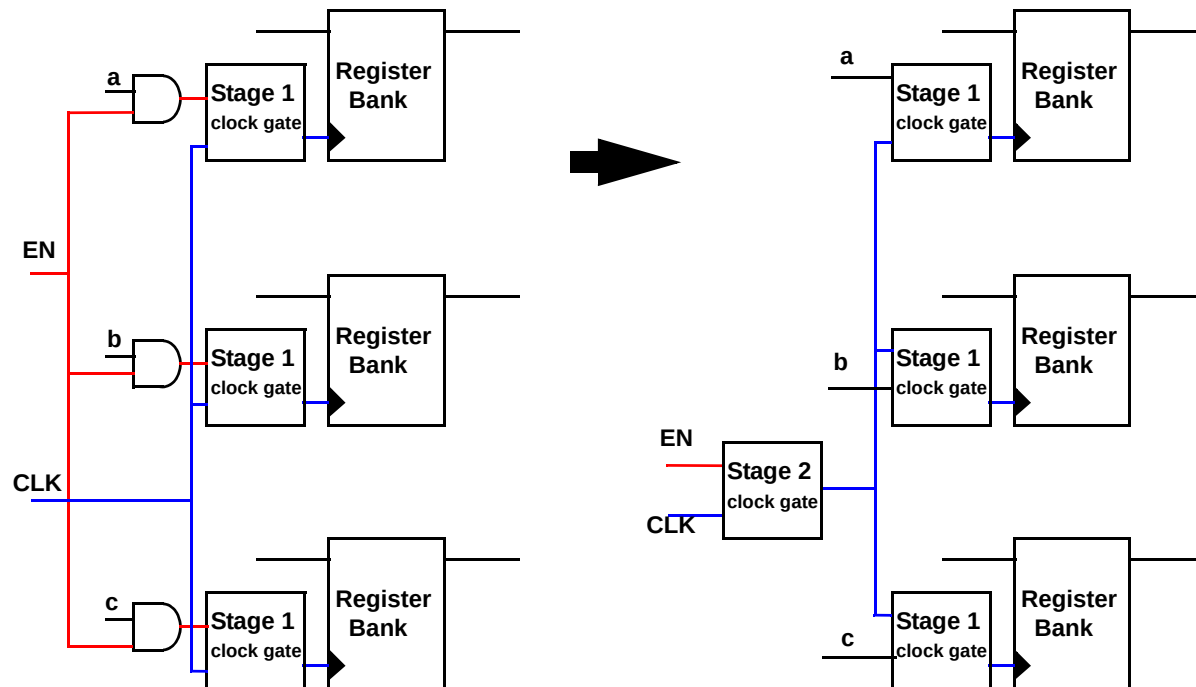
In the following example, enhanced clock gating is disabled:

```
dc_shell> read_verilog design.v
dc_shell> create_clock -period 10 -name clk
dc_shell> insert_clock_gating -regular_only
dc_shell> report_clock_gating
```

Multistage Clock Gating

When a clock-gating cell drives another or a row of clock-gating cells, it is called multistage clock gating. For additional power savings, the tool identifies common enables and factoring using another clock-gating cell as shown in [Figure 7-19](#).

Figure 7-19 Multistage Clock Gating With `set_clock_gating_style -num_stages 2`



To perform multistage clock gating, you should set the maximum number of stages for multistage clock gating by using the `-num_stages` option of the `set_clock_gating_style` command. The default value of the `-num_stages` option is 1. After setting the maximum number of stages, use either the `compile_ultra -gate_clock` or `insert_clock_gating` command to perform multistage clock gating.

However, the `compile_ultra` command performs the following additional clock-gate optimization steps during multistage clock gating:

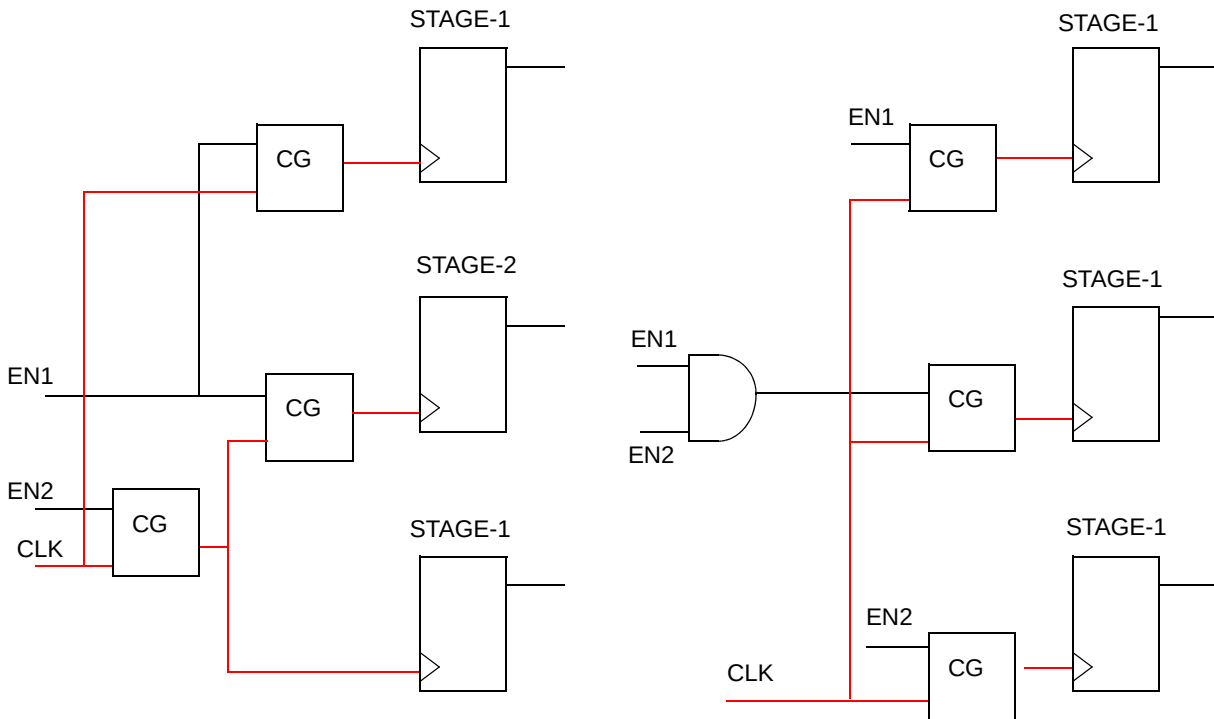
- Reconfiguring the number of clock-gating stages

If you set the `power_cg_reconfig_stages` variable to `true`, the tool reconfigures the number of clock-gating stages. The reconfiguration complies with the value of the `-num_stages` option of the `set_clock_gating_style` command. This is done only on the clock gates inserted by the tool or identified by the tool.

- Balancing the number of clock-gating stages

If you set the `power_cg_balance_stages` variable to `true`, the tool balances the number of the existing clock-gating stages across various register banks. Balanced clock-gate stages ensure uniform clock latency across register banks. [Figure 7-20](#) shows the transformation for balancing the clock-gating stages.

Figure 7-20 Balancing the Number of Stages



Multistage Clock-Gating Flow

Follow these steps to build a multistage clock-gating structure for a design that does not have clock-gating cells:

1. Set clock-gating styles and directives.

Use the `set_clock_gating_style` command to specify the clock gating stages and other clock gating conditions. You can set the number of stages for multistage clock gating as shown in the following example:

```
set_clock_gating_style -num_stages 5
```

2. Read your design.

Read in the design using a `read` command.

3. Perform multistage clock gating.

Use the `compile_ultra -gate_clock` command.

4. Report the gate elements registers.

Use the `report_clock_gating` command to get the list of cells and the `report_power` command to see the design power after the multistage clock gating.

The following is an example script to perform multistage clock gating using the `compile_ultra -gate_clock` command:

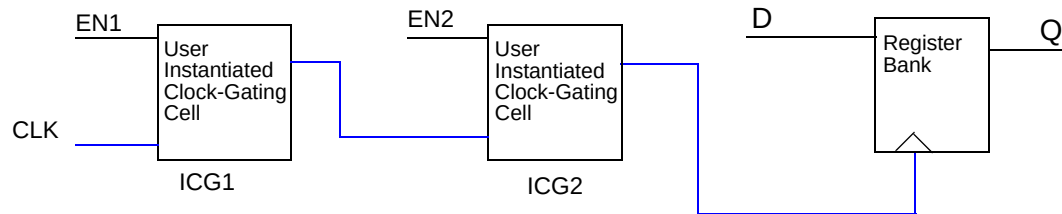
```
# set the target library and the link library

dc_shell> set_clock_gating_style -num_stages 5
dc_shell> read_verilog design.v
dc_shell> create_clock -name clk -period 10
dc_shell> compile_ultra -gate_clock
dc_shell> report_clock_gating -verbose -gating_elements \
    -gated -multi_stage
dc_shell> report_power
```

Clock Gate Merging

When your design has multiple clock-gating cells as shown in [Figure 7-21](#), the Power Compiler tool can merge two clock-gating cells, into one clock-gating cell.

Figure 7-21 Two Integrated Clock-Gating Cells

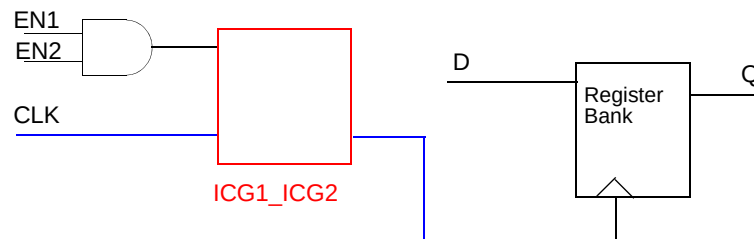


Example 7-2 shows how to insert, identify, and merge ICG1 and ICG2 clock-gating cells shown in Figure 7-22:

Example 7-2 Example to Insert, Identify, and Merge Clock-Gating Cells

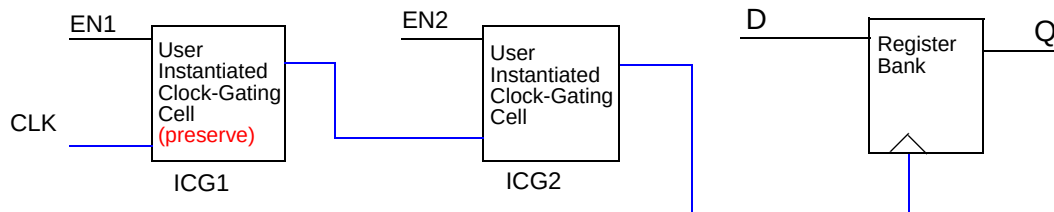
```
dc_shell> set power_cg_auto_identify true
dc_shell> set power_cg_reconfig_stages true
dc_shell> set_clock_gating_style -positive_edge_logic {integrated}\
    -num_stages 1
dc_shell> compile_ultra -gate_clock
```

Figure 7-22 Two Integrated Clock-Gating Cells Merged by AND Operation of the Enable Inputs



In Figure 7-23, one of the clock-gating cells in Figure 7-21 is set for preserve.

Figure 7-23 One Integrated Clock -Gating Cell with Preserve

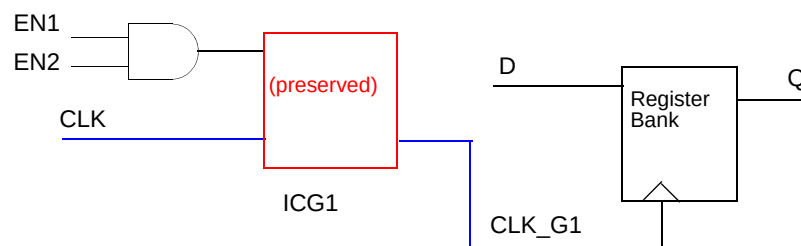


Example 7-3 shows how to insert, identify, preserve the ICG1 cell, and merge clock-gating cells as show in Figure 7-24. The preserved clock gate retains its name.

Example 7-3 Example to Insert, Identify, Preserve, and Merge Clock-Gating Cells

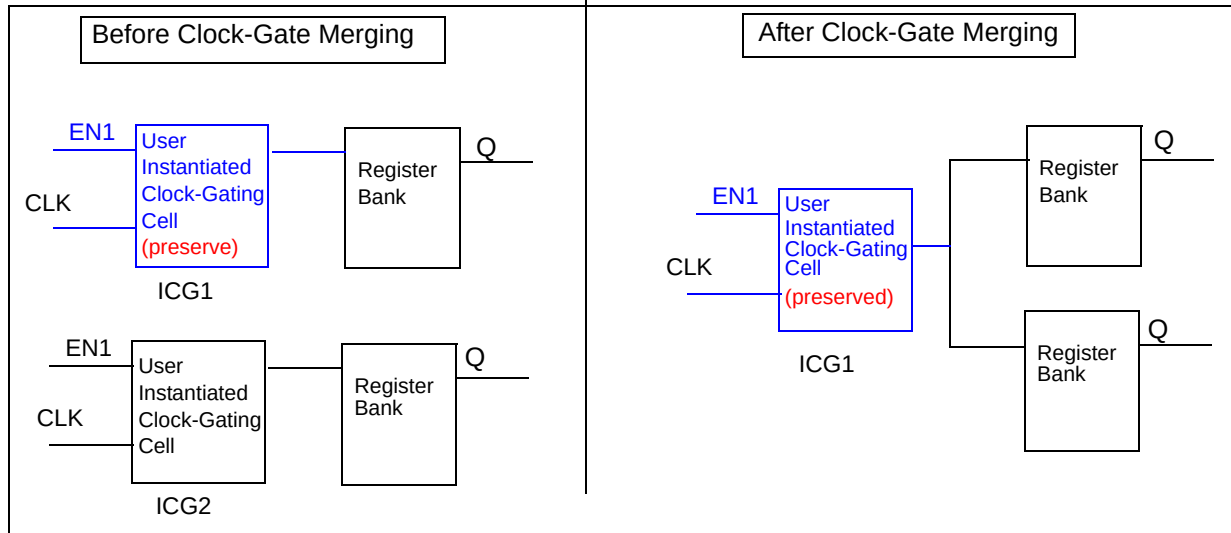
```
dc_shell> set power_cg_reconfig_stages true
dc_shell> set_clock_gating_style -positive_edge_logic {integrated} \
    -num_stages 1
dc_shell> identify_clock_gating
dc_shell> set_preserve_clock_gate [get_cells ICG1]
dc_shell> compile_ultra -gate_clock
```

Figure 7-24 Merged Clock-Gating Cells With Preserved Cell Name Remaining



In [Figure 7-25](#), one of the clock gates to be merged has the `pwr_cg_preservation_type` attribute set, and the other does not. The Power Compiler tool merges the clock gates by preserving the cell that has the `pwr_cg_preservation_type` attribute set.

Figure 7-25 Merging Clock Gates When One of the Identified Clock-Gating Cell Does Not Have the `pwr_cg_preservation_type` Attribute Set



Placement-Aware Clock Gating in Design Compiler Graphical

In the Synopsys physical guidance flow, the Design Compiler Graphical tool can restructure the connection between the integrated clock-gating cell and the registers that it drives so that the clock-gating cell and the registers can be placed close to each other. This restructuring is used by the IC Compiler tool during placement optimization, which improves the overall timing and area of the design.

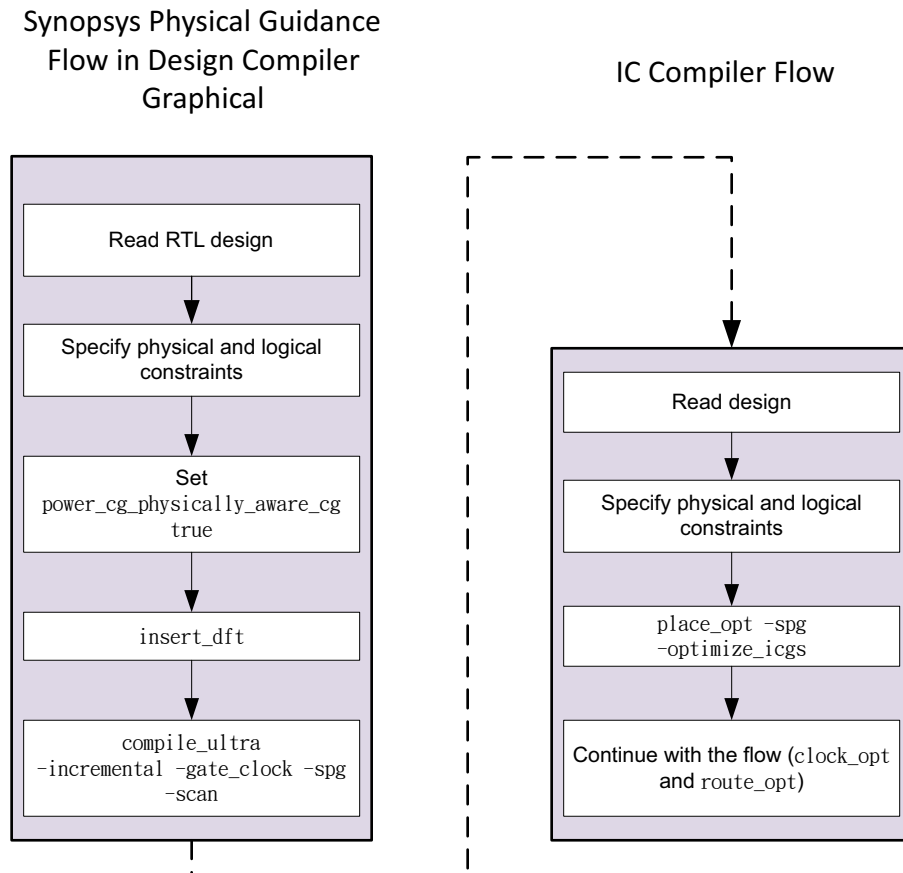
To enable the restructuring of the integrated clock-gating cell and the registers, set the `power_cg_physically_aware_cg` variable to `true`. The default is `false`. When the `power_cg_physically_aware_cg` variable is set to `true`, the annotation of clock-gate latency values is disabled during compile. Any new, tool-inserted clock gates do not have any latency annotation. This also means any latency values set using the `set_clock_latency` command are unused, and the `apply_clock_gate_latency` command has no effect. However, any previously annotated clock-gate latency values on clock-gating cells are left unchanged.

[Figure 7-26](#) shows the Synopsys physical guidance flow for placement-aware clock-gating in the Design Compiler Graphical and IC Compiler tools.

Note:

When placement-aware clock gating is enabled, clock-gating identification is performed during the `compile_ultra` command to obtain a better correlation with the IC Compiler tool. This is independent of the value of the `power_cg_auto_identify` variable.

Figure 7-26 *Synopsys Physical Guidance Flow for Placement-Aware Clock Gating in Design Compiler Graphical and IC Compiler*



Clock Gating Multibit Registers

The Power Compiler tool supports insertion of clock-gating cells on multibit registers. The enable pins for all the registers must be the same for clock gating to occur. All the clock-gating commands are supported for multibit registers.

To enable clock gating on multibit registers, set the `hdl_infer_multibit` variable to `default_all`. For further details on the multibit flow, see the *Multibit Register Synthesis and Physical Implementation Application Note*.

To set the maximum fanout value for the clock-gating cells, use the `set_clock_gating_style -max_fanout` command. When calculating the `max_fanout` value for multibit registers, use the desired register count as the fanout value.

For example, if you specify `set_clock_gating_style -max_fanout 4`, the `compile_ultra -gate_clock` command inserts a clock-gating cell that can drive up to 4 registers whether they are 4 single-bit or 4 multibit registers or a combination of the two types of registers.

It is important to note that while `max_fanout` is calculated as the actual load on the clock-gating cell, the `min_bitwidth` value is the minimum number of bits to either gate or ungate whether it is a multibit register or not. For example, if you set the `min_bitwidth` value to 3, you can clock gate four single-bit registers if they share the same clock and enable lines. The multibit mapping feature converts the four single-bit registers into a single 4-bit multibit register. Both configurations—the four single bit registers or the single 4-bit multibit register—satisfy the minimum bitwidth setting of three.

The `report_clock_gating` command generates a report that includes details of the decomposition of multibit registers, as shown in [Example 7-12 on page 7-86](#).

Note:

Power Compiler does not support XOR self-gating on multibit registers.

For more information about multibit optimization, see the *Multibit Register Synthesis and Physical Implementation Application Note*.

Performing Clock-Gating on DesignWare Components

Power Compiler provides the ability to perform clock gating on DesignWare components instead of treating them as black box cells. The `compile_ultra -gate_clock` command performs clock gating on DesignWare components, by default.

You can use the `insert_clock_gating` command to insert clock gates on DesignWare components by setting the `power_cg_designware` variable to `true`. The default is `false`.

The following example script performs clock gating on DesignWare components:

```
set power_cg_designware true
set target_library [list my_lib.db cg_integ_pos.db]
set synthetic_library dw_foundation.sldb
set link_library [list "*" my_lib.db
dw_foundation.sldb cg_integ_pos.db]
set_clock_gating_style -minimum_bitwidth 1 -sequential_cell latch \
    -positive_edge_logic {integrated:CGLP} # Optional
```

```
read_verilog cpurd_fifo.v
write -format verilog -hierarchy -output elab.v
compile_ultra -gate_clock
insert_dft
write -format verilog -hierarchy -output comp.v
```

You can view the DesignWare clock-gated registers using the `report_clock_gating -gated` command. The DesignWare clock gates are designated by a (*) in the report.

Reporting Command for Clock Gates

The `report_clock_gating` command reports the clock-gating cells and the registers with and without clock-gating signals, in the current design. To see the dynamic power savings because of clock-gate insertion, use the `report_power` command.

The `report_clock_gating` Command

The following examples are the output of the `report_clock_gating` command. If you use the `report_clock_gating` command without any option, the summary of the clock-gating elements in the current design is printed, as shown in [Example 7-4](#).

Example 7-4 Clock-Gating Report Using Default Settings

```
dc_shell> report_clock_gating
*****
Report : clock_gating
Design : low_design
Version: ...
Date   : ...
*****
```

Clock-Gating Summary

Number of Clock gating elements	1
Number of Gated registers	4 (66.67%)
Number of Ungated registers	2 (33.33%)
Total number of registers	6

[Example 7-5](#) shows an example report using the `-gating_elements` option.

Example 7-5 Clock-Gating Report Using the -gating_elements Option

```
dc_shell> report_clock_gating -gating_elements

*****
Report : clock_gating
        -gating_elements
Design : low_design
Version: ...
Date   : ...
*****

-----
                        Clock-Gating Cell Report
-----

Clock Gating Bank : clk_gate_out1_reg (ss_hvt_0v70_125c: 0.7)
-----

STYLE = latch, MIN = 2, MAX = unlimited, HOLD = 0.00, OBS_DEPTH = 5

INPUTS :
    clk_gate_out1_reg/CLK = clk
    clk_gate_out1_reg/EN  = N6
    clk_gate_out1_reg/TE  = test_se

OUTPUTS :
    clk_gate_out1_reg/ENCLK = net107

Clock Gating Bank : sub/clk_gate_out_reg (ss_hvt_1v08_125c: 1.08)
-----

STYLE = latch, MIN = 2, MAX = unlimited, HOLD = 0.00,
OBS_DEPTH = 5

INPUTS :
    sub/clk_gate_out_reg/CLK = n22
    sub/clk_gate_out_reg/EN  = N6
    sub/clk_gate_out_reg/TE  = test_se

OUTPUTS :
    sub/clk_gate_out_reg/ENCLK = net95
-----
```

[Example 7-6](#) shows an example report using the -ungated, -gated, -gating_elements, and -verbose options. A table is created to display all the ungated and gated registers in your current design.

Example 7-6 Clock-Gating Report Using Gated and Ungated Elements

```

*****
Report : clock_gating
        -gating_elements
        -gated
        -ungated
        -verbose
Design  : regs
Version: ...
Date    : ...
*****
-----
                                Clock Gating Cell Report
-----
Clock Gating Bank : clk_gate_C7
-----
      STYLE = none, MIN = 3, MAX = 2048, HOLD = 0.00, SETUP = 1.30,
OBS_DEPTH = 5
  TEST INFORMATION :
    OBS_POINT = NO, CTRL_SIGNAL = scan_enable, CTRL_POINT = none
  INPUTS :
    clk_gate_C7/CLK = clk
    clk_gate_C7/EN = xi
  OUTPUTS :
    clk_gate_C7/ENCLK = xclk
  RELATED REGISTERS :
    q4_reg[3]
    q4_reg[2]
    q4_reg[1]
    q4_reg[0]
-----
                                Gated Register Report
-----


| Clock Gating Bank | Gated Register |
|-------------------|----------------|
| clk_gate_C7       | q4_reg[0]      |
|                   | q4_reg[1]      |
|                   | q4_reg[2]      |
|                   | q4_reg[3]      |
| clk_gate_q3_reg   | q3_reg[0]      |
|                   | q3_reg[1]      |
|                   | q3_reg[2]      |
|                   | q3_reg[3]      |


-----
                                Ungated Register Report
-----


| Ungated Register | Reason               | What Next ? |
|------------------|----------------------|-------------|
| q1_reg           | Min bitwidth not met |             |


```

q2_reg	Min bitwidth not met
q5_reg	Min bitwidth not met

Clock Gating Summary	
Number of Clock gating elements	6
Number of Gated registers	16 (72.73%)
Number of Ungated registers	6 (27.27%)
Total number of registers	22

Example 7-7 Clock-Gating Report Using the -ungated Option

```
*****
Report : clock_gating
        -ungated
Design : rtl
Version: ...
Date   : ...
*****
```

Ungated Register Report		
Ungated Register	Reason	What Next?
q1_reg[1]	Always enabled register	-
q1_reg[0]	Always enabled register	-

Clock Gating Summary	
Number of Clock gating elements	0
Number of Gated registers	0 (0.00%)
Number of Ungated registers	4 (100.00%)
Total number of registers	4

[Example 7-8](#) shows the report generated when you use the `-ungated` option. The report shows the specific registers that are not clock-gated and the reason for not gating them, and the steps to follow to clock-gate the registers.

Example 7-8 Clock-Gating Report Using the `-ungated` and `-nosplit` Options

```
*****
Report : clock_gating
        -ungated
        -nosplit
Design  : my_design
Version: I-2013.12
Date    : Tue Nov  5 09:05:54 2013
*****
```

Ungated Register Report		
Ungated Register	Reason	What Next?
y_reg[5]	Suitable enable signal has been excluded	Check your CG enable exclusion
constraints		
y_reg[1]	Min bitwidth not met	-
y_reg[0]	User excluded register	-
y_reg[9]	Suitable enable signal has been excluded	Check your CG enable exclusion
constraints		
y_reg[2]	Min bitwidth not met	-

Clock Gating Summary		
Number of Clock gating elements	0	
Number of Gated registers	0 (0.00%)	
Number of Ungated registers	10 (100.00%)	
Total number of registers	10	

[Example 7-9](#) shows a report generated with the `-multi_stage` and `-no_hier` options for a hierarchical multistage clock gated design. A multistage clock gate is a clock-gating cell that is driving another clock-gating cell. The report shows three clock-gating elements, eight gated and no ungated registers at the top level. Two of the three clock gates are multistage, and their average fanout is 1.0, indicating that the clock path consists of a chain of three

clock gates. There is one gated module in addition to the eight gated registers. The eight registers have three stages on their clock path, but the module has only two, bringing the average number of stages to $2.9 = ((8*3 + 2*1)/9)$.

Example 7-9 Clock-Gating Report Using the -no_hierarchy and -multi_stage Options

```
*****
Report : clock_gating
       -no_hier
       -multi_stage
Design : regs
Version: X-2005.09
Date   : August 26, 2016 11:16 am
*****
```

Clock Gating Summary

Number of Clock gating elements	6
Number of Gated registers	16 (72.73%)
Number of Ungated registers	6 (27.27%)
Total number of registers	22
Number of multi-stage clock gates	2
Average multi-stage fanout	2.0
Number of gated cells	16
Maximum number of stages	3
Average number of stages	2.2

Example 7-10 Clock-Gating Report Using the -style Option

Clock Gating Style Report

Clock Gating Style 1 : (3 clock gates)

```
STYLE
  sequential_cell latch
  minimum_bitwidth 2
  enhanced_min_bitwidth 4
  positive_edge_logic integrated:TLATNTSCAX12MTH
  negative_edge_logic or
  control_point before
  control_signal scan_enable
  observation_point false
```

```

num_stages 2

CLOCK GATES
  clk_gate_out1_reg
  sub/clk_gate_out_reg
  sub/r0/clk_gate_out_reg

```

Clock Gating Summary

Number of Clock gating elements	3
Number of Gated registers	16 (100.00%)
Number of Ungated registers	0 (0.00%)
Total number of registers	16

Use the `report_clock_gating -structure` command to get the details and a summary of the clock-gating structure.

Note:

You cannot use the `-structure` option with any other option.

[Example 7-11](#) shows the clock-gating report when you specify the `-structure` option.

Example 7-11 Clock-Gating Report Using the -structure Option

```

*****
Report : clock_gating
        -structure
Design : test
Version: G-2012.06
Date   : Mon April 16 15:01:12 2012
*****

```

Clock Gating Structure Summary

Clock	Total Registers	CG Stage	# of Clock Gates	# of Gated Cells
clka	284	1	9	285

Clock Gating Structure Details

Clock	CG Stage	Gating Element	Fanout	Latency	Gated Cells
clka	1	cg_1	2	0.000	macro_inst
		clk_gate_y_reg	132	0.000	S4/y_reg[0] S4/y_reg[1] S4/y_reg[2] S4/y_reg[22]

	S7/clock_gate_y_reg	4	0.000	S4/y_reg[23] S7/y_reg[0] S7/y_reg[1] S7/y_reg[2] S7/y_reg[3]
	S8/clock_gate_y_reg	4	0.000	S8/y_reg[0] S8/y_reg[1] S8/y_reg[2] S8/y_reg[3]
	S9/clock_gate_y_reg	4	0.000	S9/y_reg[0] S9/y_reg[3] S9/y_reg[1] S9/y_reg[2]

Clock Gating Summary	
Number of Clock gating elements	9
Number of Gated registers	285 (100.00%)
Number of Ungated registers	0 (0.00%)
Maximum number of stages	1
Total number of registers	285

[Example 7-12](#) shows the report example for a design with multibit registers, when you use the `report_clock_gating` command without any option. The report includes details about the decomposition of the clock-gated multibit registers into single-bit registers. The Actual Count column represents the count of the cells in the design. The Single-bit Equivalent column represents the register count if every multibit register is converted to its equivalent single-bit registers.

Note:

The register count in the Clock Gating Summary is based on the count of the cells in the design that is also mentioned in the Actual Count column.

Example 7-12 Clock-Gating Report for a Design With Clock-Gated Multibit Registers

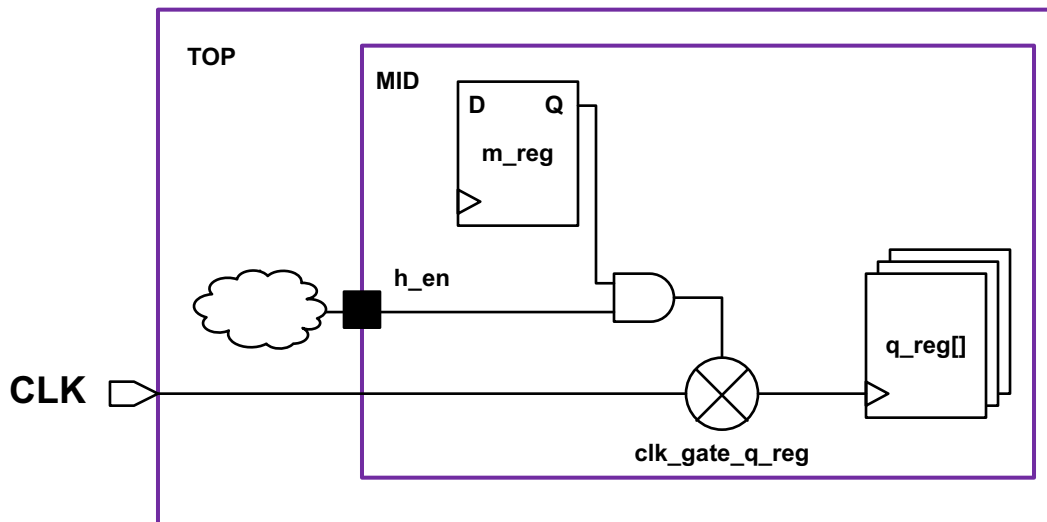
```
*****
Report : clock_gating
Design : test
Version: ...
Date   : ...
*****
```

Clock Gating Summary		
Number of Clock gating elements	1	
Number of Gated registers	1 (50.00%)	
Number of Ungated registers	1 (50.00%)	
Total number of registers	2	

Clock Gating Multibit Decomposition		
	Actual Count	Single-bit Equivalent
Number of Gated Registers		
1-bit	0	0
4-bit	1	4
Total	1	4
Number of Ungated Registers		
1-bit	1	1
4-bit	0	0
Total	1	1
Total Number of Registers		
1-bit	1	1
4-bit	1	4
Total	2	5

[Figure 7-27](#) shows the enable conditions of a clock-gating cell. The enable condition for the clock-gating cell, `clk_gate_q_reg`, is represented by the RTL invariant object names during synthesis. In this example, these objects are the sequential output pins (the Q pin coming from the `m_reg` register) and hierarchical input pins (the `h_en1` pin). These objects are put into one basic logical expression representing when the clock signal is disabled.

Figure 7-27 Example of Enable Conditions for Clock-Gating Cells



The `report_clock_gating -enable_conditions` command prints a report, as shown in [Example 7-13](#).

Example 7-13 Example of Clock Gating Report for Enable Conditions

```
*****
Report : clock_gating
        -enable_conditions
Design : top
Version: ...
Date   : ...
*****
```

Enable Conditions Report	
Clock Gating Bank	Gated Register
MID/clk_gate_q_reg	MID/q_reg
Enable condition: MID/h_en & MID/m_reg/Q	

8

XOR Self-Gating

XOR self-gating is an advanced clock-gating technique that reduces dynamic power consumption. XOR self-gating turns off the clock signal during specific clock cycles when the data in the register is unchanged.

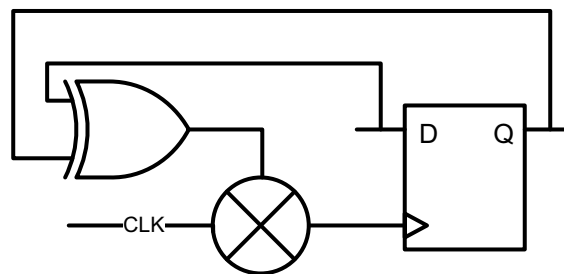
This chapter includes the following sections:

- [XOR Self-Gating Concepts](#)
- [Using XOR Self-Gating in Power Compiler](#)
- [Sharing XOR Self-Gates](#)
- [Inserting XOR Self-Gates](#)
- [Querying the XOR Self-Gates](#)
- [Reporting the XOR Self-Gates](#)

XOR Self-Gating Concepts

The XOR self-gating technique reduces the dynamic power of your design by turning off the clock signal of certain registers during specific clock cycles when the data in the register remains unchanged. With this technique, an XOR gate compares the data stored in the register with the data arriving at the data pin of the register. The XOR output controls the enable condition for gating. If the data is unchanged, the unnecessary clock cycles are gated by the output of the XOR gate. [Figure 8-1](#) shows the XOR self-gate insertion and the XOR gate that generates the enable signal.

Figure 8-1 XOR Self-Gating Cell

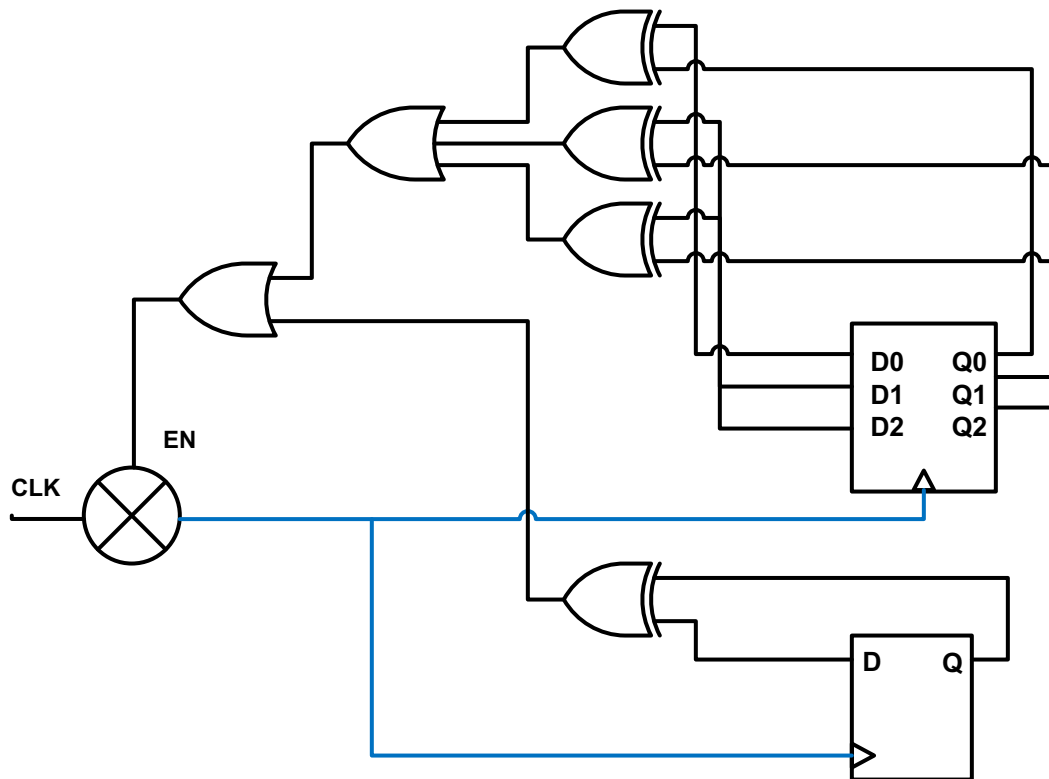


By default, the tool supports XOR self-gating only on registers that are not gated. The following are the advantages of XOR self-gating:

- You can gate registers with an enable condition that cannot be inferred from the existing logic. Therefore, these registers cannot be gated using traditional clock-gating.
- For registers that are already gated, you can use the `set_self_gating_options -interaction_with_clock_gating insert` command to allow self-gating on these registers. For these registers, the time duration that the clock signal is turned off might increase.

To minimize the area and power overhead, an XOR self-gating cell can be shared across a few registers by creating a combined enable condition with a tree of XOR gates. If the self-gated registers are driven by synchronous set or synchronous clear signals, these signals are also included in the construction of the enable signal so that the circuit remains functionally unchanged. [Figure 8-2](#) is an example of a self-gating cell that is shared across two registers (4 bits). Note that one of the self-gated registers is a multibit register and the other register is a single-bit register. The tool can also self-gate a group of multibit registers or a group of single-bit registers.

Figure 8-2 XOR Self-Gating Tree



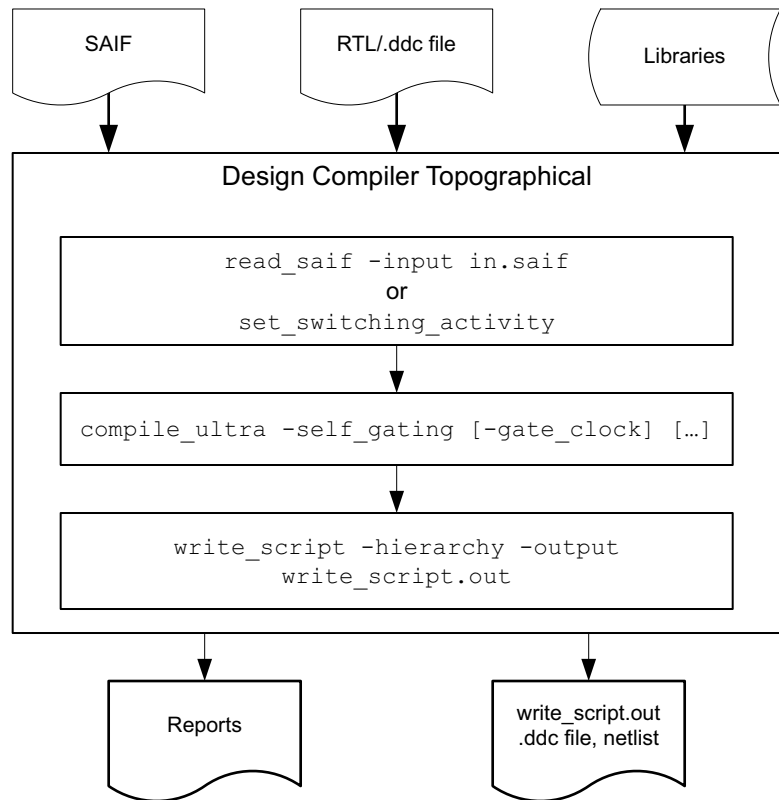
Using XOR Self-Gating in Power Compiler

In the Power Compiler tool, the XOR self-gating feature identifies those registers where the XOR self-gate insertion can potentially save dynamic power, without degrading the timing. Dynamic power is calculated using the switching activity annotated on the design. If the switching activity is not specified, the tool uses the default activity. By default, registers are grouped to create XOR self-gating banks with a minimum size of four bits and a maximum size of eight bits. The XOR self-gating cells are inserted without a hierarchical wrapper around them.

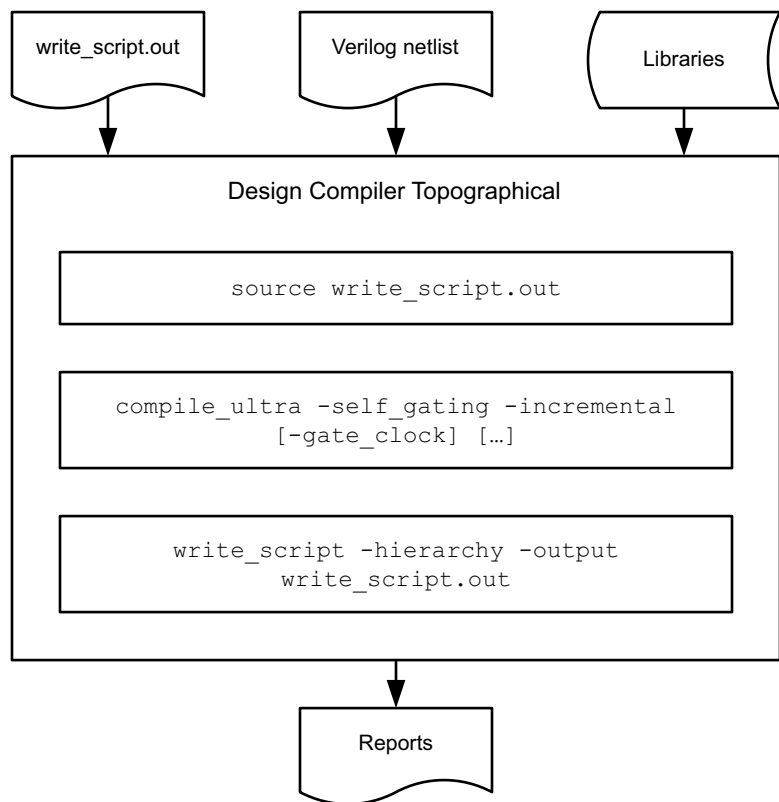
XOR Self-Gating Flows

Figure 8-3 illustrates the general flow for XOR self-gating. The general flow uses a SAIF, RTL, or .ddc file, and logic libraries.

Figure 8-3 General XOR Self-Gating Flow



When an ASCII netlist with XOR self-gating cells is read back into the Power Compiler tool, all attribute information is lost and the tool does not recognize the self-gating cells for reporting or optimization. The Power Compiler tool supports the XOR self-gating ASCII flow using the `write_script` command. As shown in [Figure 8-4](#), use the `write_script -hierarchy -output file_name` command for saving the current attributes of the design. When the design is read back into the tool, use the `source` command to source the file written by the `write_script` command. This sets all the required attributes on the design, including the self-gating cells for reporting and optimization purposes.

Figure 8-4 XOR Self-Gating ASCII Flow

Library Requirements for XOR Self-Gating

To perform XOR self-gating, the logic library should contain XOR, OR, and AND gates for the corresponding operating conditions. The integrated clock-gating cells in the library that have the following configurations are used as self-gating cells.

- Sequential cell: latch
- Control point: before
- Control signal: `scan_enable`
- Observation point: none

When the library does not contain cells with these configurations for the corresponding operating conditions, the tool does not insert XOR self-gating cells. If an integrated clock gate compatible with self-gating is specified through the `set_clock_gating_style` command, XOR self-gating uses the same integrated clock gate or the clock gate that is most similar to the one specified.

Unsupported Registers for XOR Self-Gating

The Power Compiler tool does not support the following types of sequential cells for XOR self-gate insertion:

- Level-sensitive sequential cells
- Level-sensitive scan design registers
- Master-slave flip-flops
- Retention registers
- Single-bit and multibit registers that belong to shift registers
- Multibit registers with multiple clock pins

Sharing XOR Self-Gates

If the following conditions are met, two or more registers can be gated by the same XOR self-gating cell:

- The registers belong to the same hierarchy
- The registers belong to the same clock domain
- If the synchronous global signals exist, the registers are driven by the same synchronous signals: synchronous set and synchronous clear
- If the asynchronous global signals exist, the registers are driven by the same asynchronous signals: asynchronous set and asynchronous clear

Inserting XOR Self-Gates

The XOR self-gating feature is supported only in the Design Compiler topographical mode. Use the `compile_ultra -self_gating` command to insert the XOR self-gates. The XOR self-gates are inserted using the objects specified by the `set_self_gating_objects` command and the options specified by the `set_self_gating_options` command.

Note:

Clock gate latency cannot be annotated on a self-gating cell by using the `set_clock_gate_latency` command.

For more details, see the `compile_ultra` command man page.

Specifying Objects for XOR Self-Gating

To specify objects for XOR self-gating, use the `set_self_gating_objects` command. You can specify registers, hierarchical cells, power domains, or designs for self-gating. Self-gating is performed when you run the `compile_ultra -self_gating` command.

The following command excludes the XOR self-gate to the D_OUT_reg register bank of the MID subdesign.

```
dc_shell-topo> set_self_gating_objects -exclude MID/D_OUT_reg[*]  
dc_shell-topo> compile_ultra -self_gating
```

For more details see the `set_self_gating_objects` command man page.

XOR Self-Gating the Clock-Gated Registers

By default, the Power Compiler tool does not perform XOR self-gating on clock-gated registers. If you want to perform self-gating on registers that are clock-gated, use the following command:

```
dc_shell-topo> set_self_gating_options -interaction_with_clock_gating \  
insert
```

Specifying Options for XOR Self-Gating

To specify conditions for XOR self-gating and define the interaction with clock gating, use the `set_self_gating_options` command. The `compile_ultra -self_gating` command uses these values when inserting the XOR self-gates.

The following example shows how to use the `set_self_gating_options` command to insert XOR self-gate for a minimum of two bits and a maximum of nine bits.

```
dc_shell-topo> set_self_gating_options -min_bitwidth 2 -max_bitwidth 9  
dc_shell-topo> compile_ultra -self_gating
```

When you specify the `set_self_gating_options` command with the `-interaction_with_clock_gating` option as `none`, the tool excludes the registers that are already clock gated, from XOR self-gate insertion.

For more details see the `set_self_gating_options` command man page.

Querying the XOR Self-Gates

In the current design, use the `all_self_gates` command to get a collection of self-gating cells or pins of self-gating cells.

For example, the following command returns the self-gating cells that gate registers clocked by CLK:

```
dc_shell-topo> all_self_gates -clock CLK
```

For more details, see the `all_self_gates` command man page.

Reporting the XOR Self-Gates

In the Design Compiler topographical mode, use the `report_self_gating` command to report the XOR self-gating cells. The command reports the number of registers with XOR self-gates, and optionally, information about registers without XOR self-gates in the current design.

For more details, see the `report_self_gating` command man page.

[Example 8-1](#) shows the report generated by the `report_self_gating` command when no option is specified.

Example 8-1 Report Generated by the report_self_gating Command

```
dc_shell> report_self_gating
```

```
*****
Report : self_gating
Design : my_design
Version: ...
Date   : ...
*****
```

Self-Gating Summary

Number of self gating cells	7
Number of self gated registers	50 (50.00%)
Number of registers not self-gated	50 (50.00%)
Total number of registers	100

Self Gating Multibit Decomposition

	Actual Count	Single-bit Equivalent
Number of Self Gated Registers		
1-bit	3	3
4-bit	3	12
Total	6	15
Number of Registers not Self-Gated		
1-bit	0	0
4-bit	0	0
Total	0	0
Total Number of Registers		
1-bit	3	3
4-bit	3	12
Total	6	15

Example 8-2 shows the report generated by the `report_self_gating -ungated` command. The report shows why self-gating did not occur and suggests how to get self-gating to occur on these registers.

Example 8-2 Report Generated by the report_self_gating -ungated

```
dc_shell> report_self_gating -ungated
```

```
*****
Report : self_gating
        -ungated
```

Design : my_design

Version: ...

Date : ...

----- Ungated Register Report

Ungated Register	Reason	What Next?
y_reg[9]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[8]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[7]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[6]	Self gating creates negative slack on path	Relax timing constraints on this path
y_reg[5]	Self gating creates negative slack on path	Relax timing constraints on this path

----- Self Gating Summary

Number of self-gating cells	0
Number of self gated registers	0 (0.00%)
Number of registers not self-gated	5 (100.00%)
Total number of registers	5

9

Power Optimization

Power Compiler performs additional steps to optimize your design for dynamic and leakage power.

This chapter contains the following sections:

- [Overview](#)
- [General Gate-Level Power Optimization](#)
- [Leakage Power Optimization](#)
- [Dynamic Power Optimization](#)
- [Enabling Power Optimization](#)
- [Performing Power Optimization](#)

Overview

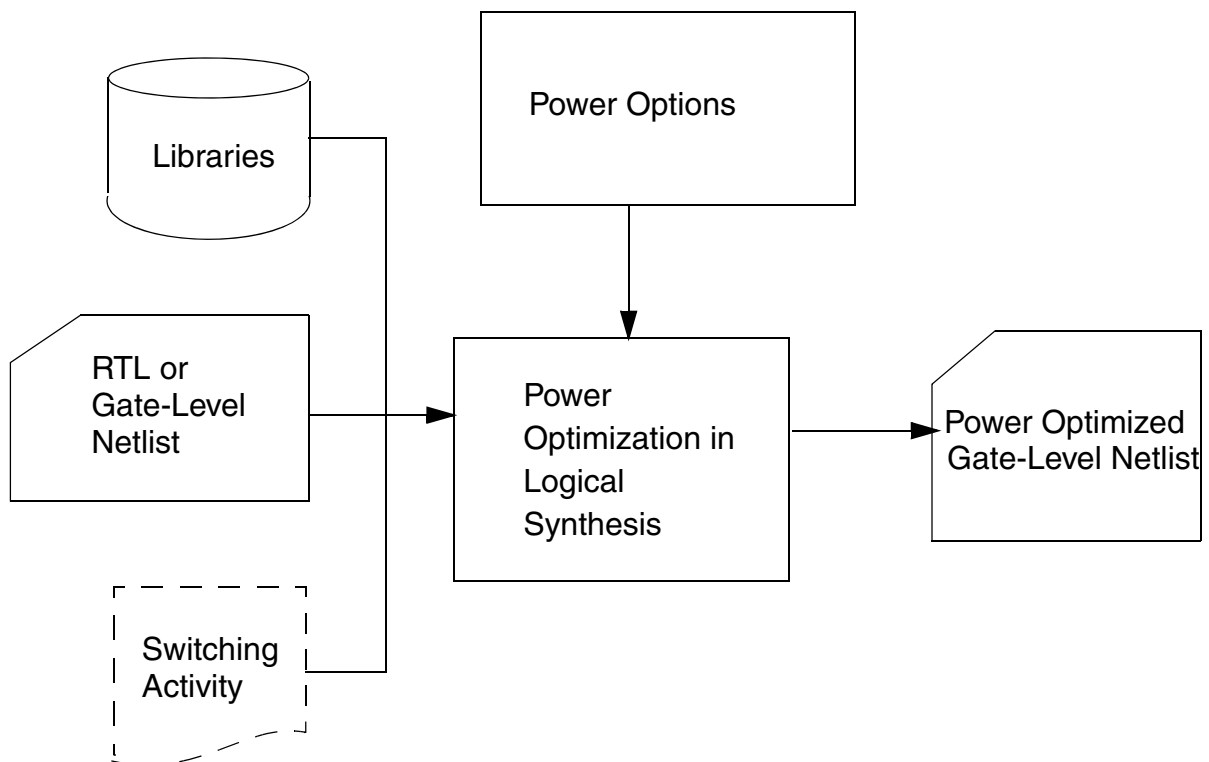
The speed of the transistor continues to improve. The most common technique used to achieve the high performance is to reduce the geometry of the transistor as well as the voltage to operate it. To maintain the speed and noise margin of the smaller transistor, the threshold voltage needs to be lowered too. Since the threshold voltage has exponential impact on the transistor leakage power, low threshold voltage transistors have high leakage power. Minimizing the leakage power is one of the major challenges to be resolved, especially in lower geometries.

In any design, there are critical and non-critical timing paths. Using a lower speed cell on non-critical path does not affect the performance of a design. A slower cell allows higher threshold voltage, which reduces leakage power dramatically. Optimizing the high speed and low speed cells on different timing paths leads to a balanced design with high performance and low leakage power.

Input and Output of Power Optimization

Figure 9-1 illustrates the flow for gate-level power optimization.

Figure 9-1 I/O Flow for Power Optimization



The inputs for gate-level power optimization are:

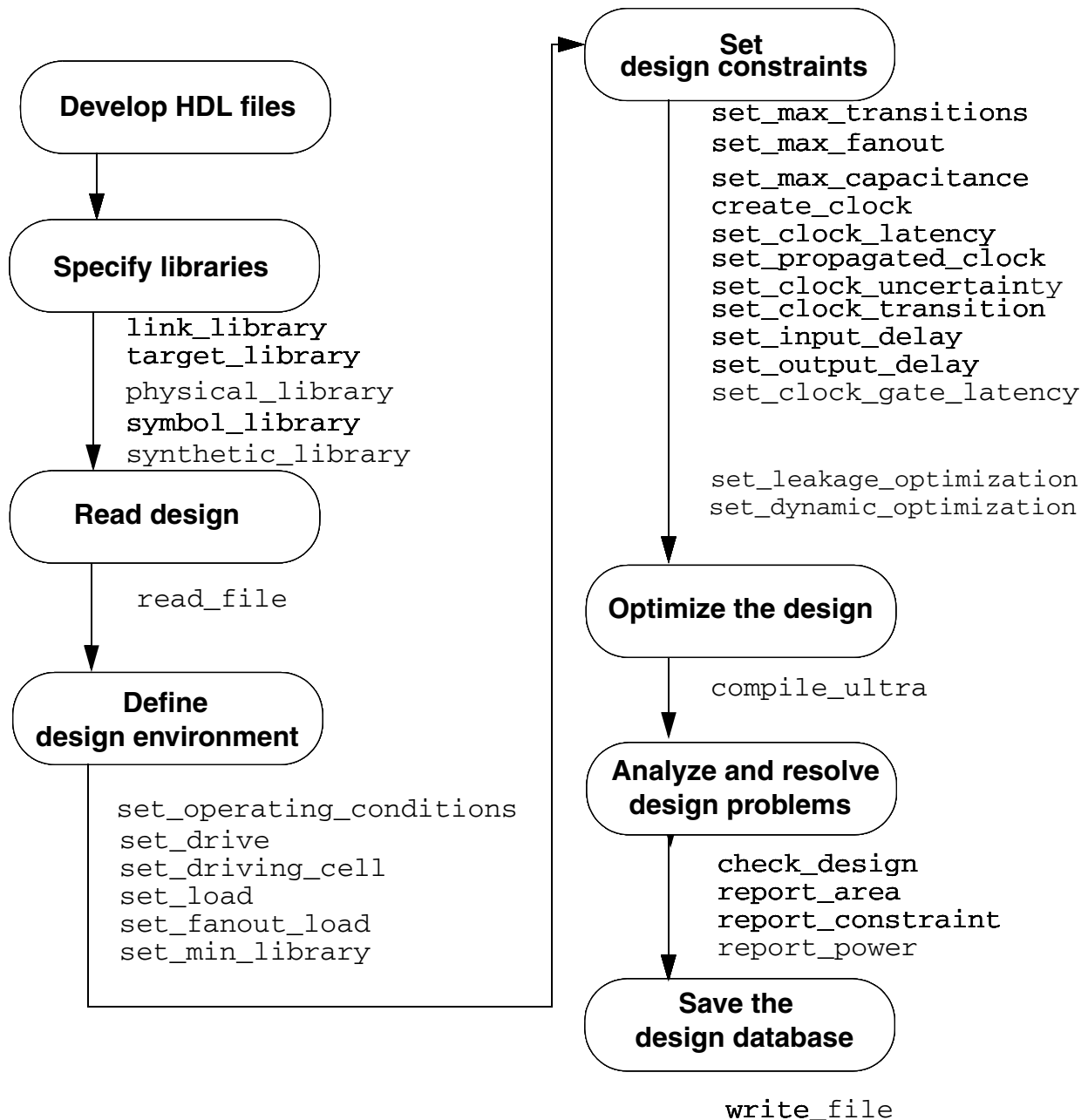
- RTL or gate-level netlist and floor plan (optional)
This netlist is not power optimized.
- Power options
Power options enable the power optimization.
- Libraries
Power Compiler selects different library cells to rebuild the netlist with the optimized power. Multivoltage threshold libraries are highly recommended for leakage optimization.
- Switching activity
This is required for dynamic and total power optimization, and is used for high accuracy in leakage optimization.

The output of gate-level power optimization is a new gate-level netlist that has optimized power. The optimization is implemented with the `compile` or `compile_ultra` commands.

Power Optimization in Synthesis Flow

Figure 9-2 shows the steps involved in power optimization in the synthesis flow.

Figure 9-2 Flow for Synthesis Power Optimization



General Gate-Level Power Optimization

To perform power optimization, Power Compiler reduces power consumption on paths with positive timing slack. The more paths in the design with positive slack, more opportunity for Power Compiler to reduce power consumption by using low-power cells. Designs with excessively restrictive timing constraints have little or no positive slack to trade for power reductions.

Designs that have black box cells, such as RAM and ROM, and customized subdesigns that have the `dont_touch` attribute, benefit from power optimization.

In Design Compiler topographical, to set the positive timing slack limit, use the `physopt_power_critical_range` variable. In the following example, Power Compiler optimizes only the timing paths with positive slack of 0.2 or more.

```
set physopt_power_critical_range 0.2
```

For more information, see the variable man page.

Leakage Power Optimization

Leakage power optimization is an additional step to timing optimization. During leakage power optimization, the tool tries to reduce the leakage power of your design without affecting the performance. To reduce the overall leakage power of the design, leakage power optimization is performed on paths that are not timing-critical. When the target libraries are characterized for leakage power and contain cells characterized for multiple threshold voltages, Power Compiler uses the library cells with appropriate threshold voltages to reduce the leakage power of the design.

Dynamic Power Optimization

Dynamic power optimization is an additional step to the timing optimization. After the optimization, your design consumes less dynamic power without affecting the performance. Dynamic power optimization requires switching activity information. Optimizing dynamic power incrementally provides better QoR and take less runtime.

Annotating the correct switching activity information, by using a SAIF file, affects the dynamic power optimization. You can annotate switching activity in the following two ways:

- Read the SAIF file

Use the `read_saif` command to read a SAIF file to annotate the switching activity information on nets, pins, ports, and cells in the current design.

- Use the `set_switching_activity` command

You can also use the `set_switching_activity` command to annotate the switching activity information.

If switching activity is not annotated, the default toggle rate is applied to the primary inputs and outputs of the black box cells. Power Compiler propagates the default toggle rate throughout the design. The propagated toggle rates are used for dynamic power optimization.

Enabling Power Optimization

Leakage power optimization is automatically enabled for all Design Compiler tools except DC Expert. When using the DC Expert tool, use the following command to enable leakage optimization:

```
set_leakage_optimization true
```

To enable dynamic optimization in all tools, use the following command:

```
set_dynamic_optimization true
```

When both leakage and dynamic power options are enabled in the DC Expert tool, the tool performs leakage power optimization.

The following example script shows the default usage model for power optimization.

```
# Specify all multivoltage threshold libraries
set_app_var target_library "hvt.db nvt.db lvt.db"
set_app_var link_library "*" $target_library

read_verilog rtl.v
link
compile_ultra
report_power
```

Note:

The `report_power` and `report_constraint` commands use state-dependent information to calculate leakage power.

Leakage Power Optimization Based on Threshold Voltage

Leakage power optimization can use single threshold voltage or multithreshold voltage libraries. However, multithreshold voltage libraries can save more leakage power.

Leakage power is very sensitive to threshold voltage. The leakage power varies from 4 to 50 times for different threshold voltages. The higher the threshold voltage, the lower the leakage power. On the other hand, timing varies from 5 to 30 percent. The lower the threshold voltage, the faster the timing. For the single-voltage library, the variance of threshold voltage and timing is of a similar magnitude.

For designs that have strict timing constraints to be met, you optimize for leakage power only on the non timing-critical paths, using the higher threshold-voltage cells from the multithreshold voltage libraries. When your design has a relatively easy-to-meet timing constraint, you might have a large number of low threshold-voltage cells in your design, resulting in higher leakage power consumption. One way to avoid this situation without having to change your target library settings is to use the `set_multi_vth_constraint` command to specify a very low percentage value for the lower threshold-voltage cells. For optimum results you should start with 1 to 5 percent of the number of cells in the design for the low threshold-voltage cells and gradually increase the percentage until the timing constraint is met. With this technique, your design meets the timing constraint with minimal leakage power consumption.

Multiple Threshold Voltage Library Attributes

To define threshold voltage groups in the libraries, use the `set_attribute` command and add the following attributes:

- Library-level attribute:
`default_threshold_voltage_group :string;`
- Library-cell-level attribute:
`threshold_voltage_group :string;`

With these attributes, the threshold voltages are differentiated by the string you specify. When the library has at least two threshold voltage groups or if you have defined threshold voltage groups for your library cells using the `set_attribute` command, the library cells are grouped by the threshold voltage. For accurate multiple threshold voltage optimization, define the threshold voltage group attributes.

The set_multi_vth_constraint Command

Use the `set_multi_vth_constraint` command to set the multithreshold voltage constraint. This command has options to specify the constraint in terms of area or number of cells of the low threshold voltage group. You can also specify whether this constraint should have higher or lower priority than the timing constraint.

The `set_multi_vth_constraint` command supports the `-type` option to specify the type of the constraint. When you specify `-type hard`, Power Compiler tries to meet this constraint, even if this results in timing degradation. When you specify `-type soft`, Power Compiler tries to meet this constraint without degrading the timing.

Note:

The `-type soft` option is supported only in Design Compiler topographical mode.

While calculating the percentage of low threshold voltage cells in the design, the tool does not consider the black box cells. To consider the black box cells in the percentage calculation, specify the `-include_blackboxes` option.

After synthesis, use the `report_threshold_voltage_group` command to see the percentage of the total design, by cell count and by area, that is occupied by the low-threshold voltage cells.

In the following example, the maximum percentage of low-threshold voltage cells in the design is set to 15 percent. When the Power Compiler tries to meet this constraint, the timing constraint is not compromised.

```
dc_shell-topo> set_multi_vth_constraint \  
                -lvth_groups {lvt svt} -lvth_percentage percentage \  
                -type soft -include_blackboxes
```

Note:

If the `set_multi_vth_constraint` command is used, it takes precedence over leakage optimization.

Analyzing the Multiple Threshold Voltage Library Cells

The Power Compiler tool supports the `analyze_library -multi_vth` command to compare the leakage power with respect to the timing characteristics of the target library cells belonging to each threshold voltage group. [Example 9-1](#) shows the report generated by the command.

Example 9-1 Report Generated by the analyze_library -multi_vth Command

```

*****
Multi-VT Library Analysis Report

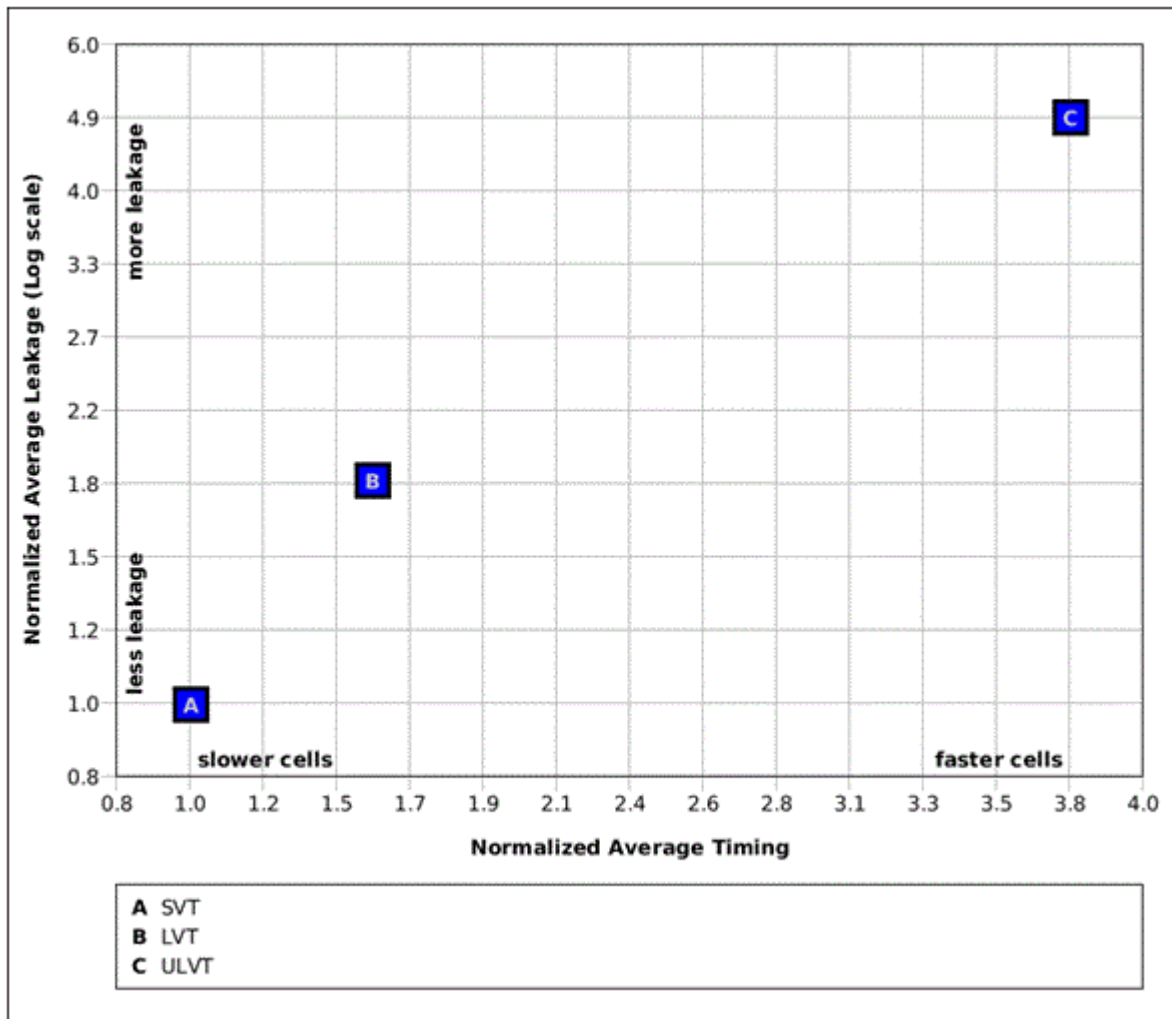
Vth Group/Library Name          Avg.          Avg.
(don't use cells/total cells)  Leakage       Timing
*****
SVT (0/998)                     1.00          1.00 (Baseline)
LVT (0/793)                     1.80          1.59
ULVT (0/998)                    4.88          3.75
*****

```

You can also generate a graphical representation of the information using the `-graph` option as shown in [Figure 9-3](#). The results of the `analyze_library` command are shown relative to a baseline library. In this figure, the baseline library is the SVT voltage threshold group. This is the library that the tool considers to have the least leakage; it is also known as an HVt (high voltage threshold group). The timing and leakage values are measured relative to this baseline. For instance, in [Figure 9-3](#), the ULVT voltage threshold group has 4.88 times more leakage than the baseline SVT group. The ULVT voltage threshold group is also 3.75 times faster in terms of delay than the SVT group. This information is provided so you can see how the tool views the performance and leakage of your voltage threshold groups relative to each other. This information is presented in a design-independent way and does not rely on any design-specific constraints.

The graph in [Figure 9-3](#) can help you determine which library to use for your optimization goals. If your design is power sensitive, you might avoid using the ULVT voltage threshold group to save power. If your design has high-performance targets, you might want to exclude the SVT group to achieve your performance goals. In this case, you sacrifice power for performance.

Figure 9-3 Graphical Output Generated by analyze_library -graph Command



Leakage Optimization for Multicorner-Multimode Designs

For multicorner-multimode designs, you can specify the leakage power optimization for specific scenarios, using the `set_scenario_options` command.

For more information about leakage power optimization for multicorner-multimode design, see [Power Optimization in Multicorner-Multimode Designs](#).

Performing Power Optimization

The `compile_ultra` command performs power optimizations, by default, along with the timing and area optimizations.

An incremental compile uses the existing netlist as a starting point for continued optimization. Usually, this ensures improvement for timing, power, and area (or for other active constraints you define). If you have a design goal that is not met (a violated constraint), a subsequent optimization session attempts to meet the violated constraint by sacrificing lower-priority design goals.

Settings for Power Optimization

The power optimization and prediction settings are used when you run the `compile_ultra` or `compile_ultra -incremental` command to perform accurate power estimation. The tool also uses these settings to get accurate post-synthesis power numbers comparable with the place-and-route numbers. Design Compiler Graphical supports IEEE 1801, also known as Unified Power Format (UPF), in the Synopsys physical guidance flow. For more details on UPF, see [UPF Multivoltage Design Implementation](#).

To enable clock-gate optimization, use the `-gate_clock` option and the `-spg` option of the `compile_ultra` command. Power Compiler inserts, modifies, or deletes clock-gating cells, except where you have set the `dont_touch` attribute on a clock-gating cell or its parent hierarchical cell.

When you enable the power prediction feature by using the `set_power_prediction` command, the Power Compiler tool performs clock tree estimation during the last phase of the `compile_ultra` command.

The `report_power` command reports the correlated power when the design is mapped to technology-specific cells. When the power prediction feature is disabled, the `report_power` command reports only the total power, static power, and dynamic power used by the design without accounting for the estimated clock-tree power.

For more details about using the low-power placement feature in multicorner-multimode designs, see [Optimizing for Dynamic Power Using Low-Power Placement](#).

Power Optimization in the Synopsys Physical Guidance Flow

The Synopsys physical guidance feature enables Design Compiler Graphical to save the physical guidance information and pass this information to IC Compiler. This section discusses the settings required for the low-power placement feature. For general details of the Synopsys physical guidance flow, see the *Design Compiler User Guide*.

Settings for Low-Power Placement

Use the following command and variable settings to enable low-power placement:

- `set_dynamic_optimization true`
- `power_low_power_placement true`

When you enable the low-power placement feature, the tool optimizes the dynamic power by shortening the net lengths of high-switching activity nets. Since the dynamic power saving is based on the switching activity of the nets, annotate the switching activity by using the `read_saif` command. Then, synthesize the design using the `compile_ultra -spg` command. [Example 9-2](#) shows how to enable and use the low-power placement feature.

Example 9-2 Enabling and Using the Low-Power Placement Feature

```
set_dynamic_optimization true
set_app_var power_low_power_placement true
read_saif -input s1.saif -instance_name inst_1
compile_ultra -spg
report_power
```

It is recommended, but not required, to read in the RTL SAIF file before optimization. If the RTL SAIF file is not available, the tool uses the defaults, a static probability of 0.5 and a toggle rate of 0.1. If you want to annotate your own values, use the `set_switching_activity` command.

10

Multivoltage Design Concepts

In multivoltage designs, the subdesign instances operate at different voltages. In multisupply designs, the voltages of the various subdesigns are the same, but the blocks can be powered on and off independently. In this user guide, unless otherwise noted, the term multivoltage includes multisupply and mixed multisupply-multivoltage designs.

This chapter contains the following sections:

- [Multivoltage and Multisupply Designs](#)
- [Library Requirements for Multivoltage Designs](#)
- [Power Domains](#)
- [Voltage Areas](#)

Multivoltage and Multisupply Designs

The logic synthesis tools support the following types of low-power designs:

- Multivoltage
- Multisupply
- Mixed multivoltage and multisupply

To reduce power consumption, multivoltage designs typically make use of power domains. The blocks of a power domain can be powered up and down, independent of the power state of other power domains (except where a relative always-on relationship exists between two power domains).

Multivoltage designs have nets that cross power domains to connect cells operating at different voltages. Some power domains can be always-on, that is, they are never powered down, while others might be always-on relative to some specific power domain. Some power domains shut down and power up independently, but might require isolation and other special cells. In general, voltage differences are handled by level shifters, which step the voltage up or down from the input side of the cell to the output side. The isolation cells isolate the power domain. Note that an enable-type level shifter can be used as isolation cells.

Library Requirements for Multivoltage Designs

To synthesize your multivoltage design using Power Compiler, the technology libraries used must conform to the Liberty syntax. It should also contain special cells such as clock-gating cells, level-shifters, isolation cells, retention registers, and always-on buffers and inverters. To support synthesis of multivoltage designs, the tool also supports multiple libraries characterized at different voltages. The following sections describe the types of cells that support multivoltage or low-power designs:

- [Liberty PG Pin Syntax](#)
- [Level-Shifter Cells](#)
- [Isolation Cells](#)
- [Requirements of Level-Shifter and Isolation Cells](#)
- [Retention Register Cells](#)
- [Power-Switch Cells](#)
- [Always-On Logic Cells](#)

Liberty PG Pin Syntax

In the traditional, non-multivoltage designs, all components of the designs are connected to a single power supply at all times. So, the technology libraries used for synthesizing such designs do not contain details of power supply and ground connections of cells because all the cells are connected to the same type of VDD and VSS.

For the synthesis of multivoltage designs, it is necessary to specify the power supplies that can be connected to specific power pins of a cell. The Liberty syntax supports the specification of power rail connection to the power supply pins of the cells. This power and ground (PG) pin information allows synthesis tool to optimize the design for power and to analyze the design behavior where multiple supply voltages are being used. For specific information about the PG pin syntax and the modeling of power supply pin connections, see the Advanced Low Power Modeling chapter in the *Library Compiler User Guide*.

For an older library that does not contain PG pins, you can convert the library into PG pin library format in Design Compiler. For more details, see [Converting Libraries to PG Pin Library Format](#).

Level-Shifter Cells

In a multivoltage design, a level shifter is required where a signal crosses from one power domain to another. The level shifter operates as a buffer with one supply voltage at the input and a different supply voltage at the output. Thus, a level shifter converts a logic signal from one voltage level to another, with a goal of having smallest possible delay from input to output.

Level-shifter cells are of three types:

- Level shifters that convert from high voltage to low voltage (H2L)
- Level shifters that convert from low voltage to high voltage (L2H)
- Level shifter that can do both, high to low and low to high conversion

PG Pin Configuration Support

In addition to the different types of voltage conversions, the Power Compiler tool supports level-shifter cells with different PG pin configurations as specified by the Library Compiler models:

- Single-rail level shifter
- Dual-rail level shifter which has two PG pins. One pin is designated as the main rail and connected to the primary power supply while the other pin is connected to a secondary rail.

- Level shifter with a feedthrough standard cell main rail (SCMR) PG pin enables shifting of always-on signals between shutdown power domains. The feedthrough SCMR PG pin is connected to the domain's primary supply, which is not part of either the power domains involved in the level shifting.
- Level shifter inside a macro cell, which is connected directly to the macro cell's input pins. This model eliminates the need to insert external level shifters.
- Enable level shifter, which performs both level shifting and isolation functions. There are two types of enable level shifters. In all the following cases, the enable pin's PG pin is connected to the isolation supply.
 - A cell that is modeled as a level shifter followed by an isolation cell. For this cell, the input data pin is powered by one supply while the enable and output pins are powered by another supply.
 - A cell that is modeled as an isolation cell followed by a level shifter cell. This cell has four terminal pins, and it can be modeled as follows:
 - input, enable, and output pins are each powered by a different supply
 - input and enable pins are powered by one supply, and the output pin is powered by another supply

For more information modeling level shifters, see the *Library Compiler User Guide*.

Isolation Cells

Isolation cells are required when a logic signal crosses from a power domain that can be powered down to a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a constant output signal when the input side is powered down.

A cell that can perform both level-shifting and isolation functions is called an enable level-shifter cell. This type of cell is used where a signal crosses from one power domain to another, where the two voltage levels are different and the first domain can be powered down.

For more information about creating and using isolation cells and enable level-shifter cells, see the *Library Compiler User Guide*.

Using Standard Cells as Isolation Cells

When your target library does not contain a complete set of isolation cells, you can use the basic 2-input AND, OR, NAND, and NOR gates as isolation cells. This flexibility allows you to use these basic cells for their usual logic as well as for isolation logic. Only the following types of basic gates can be used as isolation cells:

- 2-input AND, OR, NAND, and NOR gates
- 2-input AND, OR, NAND, and NOR gates with one of the inputs inverted

To enable this feature, you must set the `mv_use_std_cell_for_isolation` variable to `true`. You must then set the following attributes using the `set_attribute` command:

- Set the library cell-level attribute `ok_for_isolation` to `true` on the library cell.
This attribute denotes that the library cell can be used as a standard logic cell as well as an isolation cell. The following example shows how to set the `ok_for_isolation` attribute on the library cell A:

```
set_attribute [get_lib_cells lib_name/A] ok_for_isolation true
```

- Set the `isolation_cell_enable_pin` attribute to `true` on the library cell pin. This attribute specifies the pin to be used as the control pin of the isolation cell.

The following example script shows how to set the `isolation_cell_enable_pin` attribute to `true` on the `in` pin of the library cell A:

```
set_attribute [get_lib_pins lib_name/A/in] \
  isolation_cell_enable_pin true
```

Single-Rail and Dual-Rail Isolation Cells

When selecting an isolation cell for mapping, the tool automatically selects single-rail or dual-rail cells based on the isolation cell's rail information and location.

Typically, you use a single-rail isolation cell if the cell is inserted in a domain where the primary rail remains on during shutdown mode. Use a dual-rail isolation cell when the isolation cell needs to be inserted in the shutdown domain and requires a secondary rail connection, so the cell continues to be powered during shutdown mode by a backup supply.

You can create exceptions to this rule by using the `map_isolation_cell` command to restrict the availability of cells. The tool can insert a conflicting cell if the proper cell is not available. For example, if only single-rail cells are available, the tool inserts them even in a shutdown region. However, the tool does issue a warning message.

The tool checks and reports warnings for rail violations, for example, a single-rail isolation cell used in a shutdown domain or a dual-rail isolation cell used in a domain that is powered on more than the shutdown domain.

NOR-Style Isolation Cells

The Power Compiler tool automatically selects and inserts NOR-style (or clamp-to-zero) isolation cells in the shutdown domain. This isolation cell is a single-rail cell that clamps its output to ground (zero) when the domain's primary supply is turned off. The tool can also optimize dual-rail isolation cells to a single NOR-style isolation cells to reduce the power and area used.

To specify a NOR-style isolation cell, specify the isolation strategy with the `set_isolation` command and the `-isolation_supply_set` option. The `-clamp_value` option must be set to zero. For example,

```
dc_shell> set_isolation IS01 -domain PD_BLK -isolation_supply_set {} \
          -clamp_value 0 -applies_to outputs
```

Note:

Since the empty isolation supply set `{}` indicates that the isolation cell has no power supply when operating in isolation mode, the tool uses a clamp-to-zero isolation cell (or a NOR-type isolation cell).

The tool optimizes the design using NOR-type isolation cells when it can. For example,

```
dc_shell> set_isolation IS01 -domain PD_BLK \
          -isolation_supply_set {TOP_A0_SS} -clamp_value 0 \
          -applies_to outputs
```

The tool optimizes the implemented isolation cell to use the domain's primary supply, `PD_BLK.primary`, by using a NOR-type isolation cell.

For more details on modeling and using NOR-style isolation cells, see [SolvNet article 2370685](#), "Single-Rail Clamp-to-0 Nor-Type Isolation Cells."

Requirements of Level-Shifter and Isolation Cells

The following are the requirements of level-shifter and isolation cells:

- Two power supplies.
- Buffer-type and enable-type level-shifter library cells must have the `is_level_shifter` library attribute set to `true`.
- Enable-type level shifters must also have the `level_shifter_enable_pin` library attribute set on the enable pin.
- Isolation library cells must have the `is_isolation_cell` library attribute set to `true`.
- Isolation cells must have the `isolation_cell_enable_pin` library attribute set on the enable pin.

- Level shifters and isolation cells are selected by the logic synthesis tool from the target libraries. Therefore, at least one of the libraries must contain these required cells.
- Level-shifter and isolation cells can only be inserted on unidirectional ports.

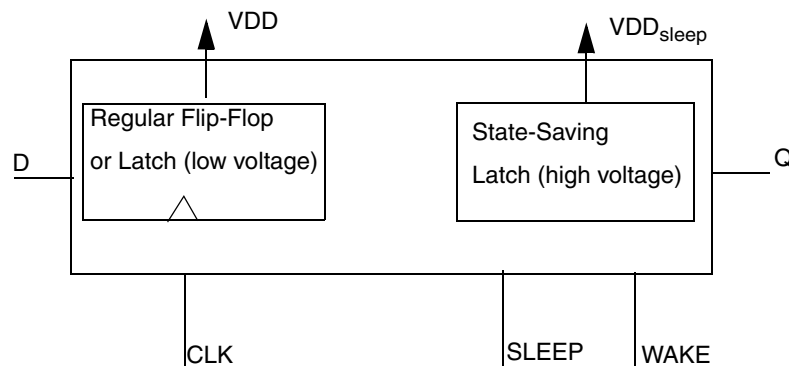
Retention Register Cells

In a design with power switching, one of the ways to save register states before power-down and restore them upon power-up is to use retention registers. These registers can maintain their state during power-down by means of a low-leakage register network and an always-on power supply. Retention cells occupy more area than regular flip-flops. These cells continue to consume power when the power domain is powered down.

Multithreshold-CMOS Retention Registers

Retention cells are sequential cells that can hold their internal state when the primary power supply is shut down and that can restore the state when the power is brought up. So the retention registers are used to save leakage power in power-down applications. During normal operation, there is no loss in performance and during power-down mode, the register state is saved. These features are possible with the addition of a state-saving latch, which holds the data from the active register. [Figure 10-1](#) shows the basic elements of the retention register.

Figure 10-1 Retention Register Components



The retention register consists of two separate elements:

- Regular Flip-Flop or Latch

The regular flip-flop or latch consists of low-threshold voltage MOS transistors for high performance

- State-Saving latch

The state-saving latch consists of a balloon circuit modeled with high-threshold voltage MOS transistors. It has a different power supply: VDD_{SLEEP}

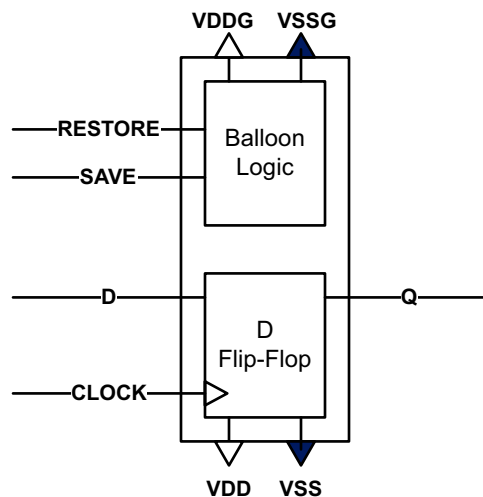
The behavior of these elements depends on the circuit mode. During active mode, the regular register operates at speed and the retention latch does not add to the load at the output. During sleep mode, the Q data is transferred to the state-saving latch, and the power supply to the flip-flop is shut off, thus eliminating the high-leakage standby power. When the circuit is activated with the wake-up signal, the data in the retention latch is transferred to the regular register for continuous operation.

Along with the separate power supplies, additional signals such as SLEEP and WAKE are required to enable the data transfer from the regular register to the state-saving latch and back again, based on the mode of operation.

Based on the application, different retention register types are available to address the clocking of the data from the register to the latch and back again. Library Compiler supports modeling of retention registers with two control pins as well as only one control pin.

[Figure 10-2](#) shows a retention register that has two control signals, save and restore, to save and restore the data. In this figure, triggering the Save pin puts the register in the active mode meaning the register works as a regular D flip-flop. Triggering the Restore pin puts it in sleep mode meaning the register works as a state-saving latch.

Figure 10-2 Two-Pin Retention Register



The Library Compiler tool also supports single-pin retention registers. For single-pin retention registers, the control pin, called a `save_restore` pin, saves and restores the state of the cell depending on the voltage state of the pin. Single-pin retention registers behave like the two-pin control retention register in [Figure 10-2](#). The only difference is that the control pin is a single pin instead of two pins. For a single-pin retention register to work like a regular latch (D flip-flop, in this case), the `save_restore` pin needs to be put into "save" mode. When the retention register is put into "restore" mode, it works like a retention cell. That is, the D-input of the register is not passed to the balloon logic. Thus, the balloon logic

of the retention register has the last known value saved in it. This value is fed to the Q-output when restore mode is enabled.

An example of a retention library cell might be defined as follows:

```
RETENTION_PIN: (save_restore, 1)
```

The disable value is 1, which means the retention cell is working in normal save mode when the `save_restore` pin is driven to 1 (high) and working in restore mode when the `save_restore` pin is driven to 0 (low).

For a UPF file specification, you need to define the retention register control in the UPF file as follows to make this work correctly:

```
set_retention_control PD1_RFF -domain PD1 -save_signal {SRPG1 high} \
-restore_signal {SRPG1 low}
```

Power-Switch Cells

In a design with power switching, the power-switch cells provide the supply power for cells that can be powered down. The library description of a power-switch cell specifies the input signal that controls power switching, the pin or pins connected to the power rail, and the pin or pins that provide the virtual or switchable power.

There are two types of power-switch cells, the header type and the footer type. A header type power switch connects the power rail to the power supply pins of the cells in the power-down domain. A footer type power switch connects the ground rail to the ground supply pins of the cells in the power-down domain.

For more information about creating power-switch cells, see the *Library Compiler User Guide*.

Always-On Logic Cells

Multivoltage designs can contain some power domains that can be shut down during the operation of the design. These are also called power-down domains. In some of the power-down domains, logic cells need to remain powered on even when the power domain is shut down. Such cells are called always-on cells. The control signals of the always-on cells should also be powered on when the power domain is shut down. These control signal paths are called always-on paths.

The always-on cells can be of two types:

- Single Power Standard Cell

Buffers and inverters from the standard cell libraries can be used as always-on cells. For Power Compiler to use the standard cells as always-on cells, you must

- Define the power domain as a shutdown domain.
For more details on always-on logic, see [Shut-Down Blocks](#).
- Set the `always_on_strategy` attribute to `cell_type` and `single_power`.
- Dual Power Special Cell
Special cells in the target library, such as buffers and inverters with dual power, can be used for always-on logic. Power Compiler automatically infers the backup power supply for these cells based on the supply load on these cells. For more details, see [Always-On Logic](#).
For more information about always-on logic, see [Shut-Down Blocks](#).

Power Domains

Multivoltage designs contain design partitions which have specific power behavior compared to the rest of the design. A power domain is a basic concept in the Synopsys low-power infrastructure, and it drives many important low-power features across the flow.

By definition, a power domain is a logical grouping of one or more logic hierarchies in a design that share the same power characteristics, including:

- Primary voltage states or voltage range (that is, the same operating voltage)
- Process, voltage, and temperature (PVT) operating condition values (all cells of the power domain except level shifters)
- Power net hookup requirements
- Power down control and acknowledge signals, if any
- Power switching style
- Same set or subset of nonlinear delay model (NLDM) target libraries

Thus, a power domain describes a design partition, bounded within logic hierarchies, that has a specific power behavior with respect to the rest of the design.

Each power domain has a supply network consisting of supply nets and supply ports and might contain power switches. The supply network is used to specify the power and ground net connections for a power domain. A supply net is a conductor that carries a supply voltage or ground. A supply port is a power supply connection point between the inside and outside of the power domain. Supply ports serve as the connection points between supply nets. A supply net can carry a voltage supply from one supply port to another.

When used together, the power domain and supply network objects allow you to specify the power management intentions of the design.

Every power domain must have one primary power supply and one primary ground. In addition to the primary power and ground nets, a power domain can have any number of additional power supply and ground nets.

A power domain has the following characteristics:

- Name
- Level of hierarchy or scope where the power domain is defined or created
- The set of design elements that comprise the power domain
- Associated set of supply nets that are allowed to be used within the power domain
- Primary power supply and ground nets
- Synthesis strategies for isolation, level-shifters, always-on cells, and retention registers

Note:

A power domain is strictly a synthesis construct, not a netlist object. For more information about the concept of Power Domain, see the *Synopsys Multivoltage Flow User Guide*.

Shut-Down Blocks

Multivoltage designs typically have some power domains that are shut down and powered up during the operation of the chip while other power domains are always powered up. The always-on paths starting from an always-on block must connect to the specific pins of always-on cells in the power-down block. These cells can be special, dual power cells (isolation cells, enable-type level shifters, retention registers, special RAMs, and so on) or standard cells that when placed are confined to special always-on site rows within the power-down block.

Specific commands are supported by the tool can be used to specify the always-on methodology to be applied to a particular power-down block. If special cells are used, they need to be marked appropriately so that the tool can determine the always-on paths and correctly optimize these paths.

Only buffers and inverters can be used as dual-power, always-on cells. They must have two rails connections: a primary rail that is connected to a shut-down power supply, and a secondary rail that is connected to an always-on power supply.

Marking Pass-Gate Library Pins

In the current implementation, the tool has the ability to stop always-on cells from connecting to cells with pass gate inputs. An always-on buffer should not drive a gate that has pass transistors at the inputs (pass-gate). Pass-gate input cells should be driven by a standard cell in a shut-down power domain. Therefore, if your library contains any of these cells, you must mark them as pass-gates in each session.

For example, to mark the pin A of the mux cell MUX1, run the following command as part of a Design Compiler script:

```
set_attribute [get_lib_pins lib_name/MUX1/A] pass_gate true
```

Voltage Areas

Corresponding to the power domains of logic synthesis, you define *voltage areas* in physical synthesis as placement areas for the cells of the power domains. Except for level shifter cells, all cells in a voltage area operate at the same voltage.

There must be an exact one-to-one relationship between logical power domains and physical voltage areas. The Design Compiler and IC Compiler tools can align the logic hierarchies of the power domains with their voltage areas with appropriate specifications. *The power domain name and the voltage area name should be identical.*

If you do not make these specifications, you are responsible for ensuring that the logic hierarchies are correctly aligned, as well as being correctly associated with the appropriate operating conditions.

A voltage area is the physical implementation of a power domain. A voltage area is associated with a power domain in a unique, tightly bound, one-to-one relationship. A voltage area is the area in which the cells of specific logic hierarchies are to be placed. A single voltage area must correspond to another single power domain, and vice versa. The power domains of a design are defined first in the logical synthesis phase and then the voltage areas are created in the physical implementation phase, in Design Compiler topographical mode or in IC Compiler. The information that pertains to logic hierarchies, which belongs to a voltage area boundary is derived from a corresponding power domain. Also, all the cells that belong to a given voltage area have the power behavior described by the power domain characteristics.

For more information, see the command man page.

11

UPF Multivoltage Design Implementation

This chapter describes multivoltage design concepts and the use of the IEEE 1801 also known as Unified Power Format (UPF), to synthesize your multivoltage designs in Power Compiler. This chapter describes specifying your power intent in the UPF file, reading the UPF file in Power Compiler, and using commands supported in UPF mode in the following sections:

- [Multivoltage Design Flow Using UPF](#)
- [Power Intent Concepts](#)
- [Defining Power Intent in UPF](#)
- [Creating Power Domains](#)
- [Creating Supply Ports](#)
- [Creating Supply Nets](#)
- [Connecting Supply Nets](#)
- [Specifying Supply Sets](#)
- [Refining Supply Sets](#)
- [Always-On Logic](#)
- [Specifying Level-Shifter Strategies](#)
- [Specifying Isolation Strategies](#)

- [Setting UPF Attributes on Ports and Hierarchical Cells](#)
- [Specifying Retention Strategies](#)
- [Creating Power Switches](#)
- [Power State Tables](#)
- [Support for Well Bias](#)
- [Inserting Power Management Cells](#)
- [Reviewing the UPF Specifications](#)
- [Examining and Debugging UPF Specifications](#)
- [Writing the Power Information](#)
- [Writing and Reading Verilog Netlists With Power and Ground Information](#)
- [Golden UPF Flow](#)
- [Reporting Commands for the UPF Flow](#)
- [UPF-Based Hierarchical Multivoltage Flow Methodology](#)

Multivoltage Design Flow Using UPF

The Unified Power Format (UPF) is a standard set of Tcl-like commands used to specify the low-power design intent for electronic systems. UPF provides the ability to specify the power intent early in the design process. Also, UPF supports the entire design flow. For more information about the low-power flow and the various Synopsys tools that support UPF, see the *Synopsys Multivoltage Flow User Guide*.

To synthesize the multivoltage design, the recommended method is to use the top-down approach. With your power intent defined in the UPF file, follow these steps to synthesize your multivoltage design:

1. Read your RTL file.
2. Read the power definitions for your multivoltage design using the `load_upf` command.

In the UPF flow, the RTL file cannot have power definitions. Power Compiler issues an error message if it encounters power definitions in the RTL file. All the power definitions must be specified in the UPF file. The UPF file can be used for synthesis, simulation, equivalence checking, and sign off.

By default, the `load_upf` command executes the commands in the associated UPF file in the current level of hierarchy. If the identifiers do not adhere to the naming rules specified in the UPF standard, the following error message is issued.

```
Error:Symbol symbol_name violates the UPF naming conventions (UPF-200.
```

The Design Compiler commands and variables, and the UPF commands and variables, defined in the UPF file, share the same namespace. While executing the `load_upf` command, the tool checks for namespace conflicts for the commands and variables already defined, and those in the UPF file being read.

For more information, see [Name Spacing Rules for UPF Objects and Attributes](#).

If you have modified the UPF file after reading it, you can use the `remove_upf` command to remove the UPF constraints. However, you cannot use the `remove_upf` command after synthesizing the design or if you read a synthesized design.

After updating or removing a UPF file, use the `load_upf` command to reload the file.

Note:

The Design Vision GUI supports the Visual UPF dialog box, which is accessible from the Power menu. Using the Visual UPF dialog box, you can define the power domains, their supply network, connections with other power domains, and relationships with elements in the design hierarchy.

For more information see [Defining the Power Intent in the GUI](#).

3. Specify the set of target libraries to be used.

Your target library must comply with the power and ground pin Liberty library syntax. The target library should also support special cells such as isolation cells and retention registers.

For more details on the target library requirement for multivoltage implementation see [Library Setup for Power Optimization](#). For additional information about the PG pin Liberty library syntax, see the Advanced Low-Power Modeling chapter in the *Library Compiler User Guide*.

4. Use the `set_operating_conditions` command to set the operating condition on the top level of the design hierarchy, and to derive the process and temperature conditions for the design.

Use the `set_voltage` command to set the current operating voltage value for the power and the ground supply nets.

5. Specify your power optimization requirements.

When you use any of the power optimization constraints in the Design Compiler topographical technology, the tool also enables power prediction using the clock tree estimation. For more details about power prediction, see [Performing Power Correlation](#).

6. Compile your multivoltage design using the `compile_ultra` command.

Note:

When you synthesize your design for the first time using Design Compiler topographical mode, use the `compile_ultra -check_only` command. The `-check_only` option checks your design and the libraries for all the data that is required by the `compile_ultra` command to successfully synthesize your design. For more details, see the *Design Compiler User Guide*.

7. Use the `check_mv_design` command to check for multivoltage violations in your design.

The command checks your design for inconsistencies in your design and the target libraries, and violations related to power management cells and their strategies. Use the `-verbose` option to get the details of the violations. The `-max_messages` option controls the number of violations being reported. For more details, see the command man page.

To identify the multivoltage inconsistencies, use the MV Advisor feature of the Design Vision GUI. For more information, see [Examining and Debugging UPF Specifications](#).

8. Write the synthesized design using the `write -format` command. When you write the design in the ASCII format, use the `change_names` command before you write out the design.

To generate the multivoltage reports, use the various reporting commands such as `report_power_domain`. For more details on multivoltage reporting commands, see [Reporting Commands for the UPF Flow](#).

9. Use the `save_upf` command to save the updated power constraints in another UPF file.

After completing the synthesis process, the UPF file written by Design Compiler is used as input to the downstream tools, such as IC Compiler, PrimeTime or PrimeTime PX, and Formality. This file is similar to the one read into Design Compiler but with the following additions:

- A comment on the first line of the UPF file generated by Design Compiler. An example is as follows:

```
#Generated by Design Compiler(H-2013.03) on Wed Feb 20 14:26:58 2011
```
- Explicit power connections to special cells such as level-shifter cells and dual supply cells.
- Any additional UPF commands that were specified at the command prompt in the Design Compiler session.

If you have specified UPF commands at the Design Compiler command prompt during synthesis, update the UPF file along with your RTL design with these commands. Without this update to the UPF file, Formality does not verify the design successfully.

An alternative method to maintain the UPF power intent of the design is called the golden UPF flow. In this method, the original UPF file that you specify is used throughout the synthesis, physical implementation, and verification steps along with supplemental UPF files generated by the tools. For more information, see [Golden UPF Flow](#).

Power Intent Concepts

The UPF language provides a way to specify the power requirements of a design, but without specifying explicitly how those requirements are implemented. The language specifies how to create a power supply network to each design element, the behavior of supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. It does not contain any placement or routing information. The UPF specification is separate from the RTL description of the design.

In the UPF language, a *power domain* is a group of elements in the design that share a common set of power supply needs. By default, all logic elements in a power domain use the same primary supply and primary ground. Other power supplies can be defined for a power domain as well. A power domain is typically implemented as a contiguous *voltage area* in the physical chip layout, although this is not a requirement of the language.

Each power domain has a *scope* and an *extent*. The *scope* is the level of logic hierarchy designated as the root of the domain. The *extent* is the set of logic elements that belong to the power domain and share the same power supply needs. The scope is the hierarchical level at which the domain is defined and is an ancestor of the elements belonging to the

power domain, whereas the extent is the actual set of elements belonging to the power domain.

Each scope in the design has *supply nets* and *supply ports* at the defined hierarchical level of the scope. A *supply net* is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net that spans more than one power domain is said to be “reused” in multiple domains. A *supply port* is a power supply connection point between two adjacent levels of the design hierarchy, between the parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

A *supply set* is an abstract collection of supply nets, consisting of two supply functions, power and ground. A supply set is *domain-independent*, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can physically implemented (placed and routed), its supply sets must be *refined*, or associated with actual supply nets.

A *supply set handle* is an abstract supply set created for a power domain. By default, a power domain has supply set handles for the domain's primary supply set, a default isolation supply set, and a default retention supply set. These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can be physically implemented, its supply set handles must be *refined*, or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

A *power switch* (or simply *switch*) is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals. A *power state table* lists the allowed combinations of voltage values and states of the power switches for all power domains in the design.

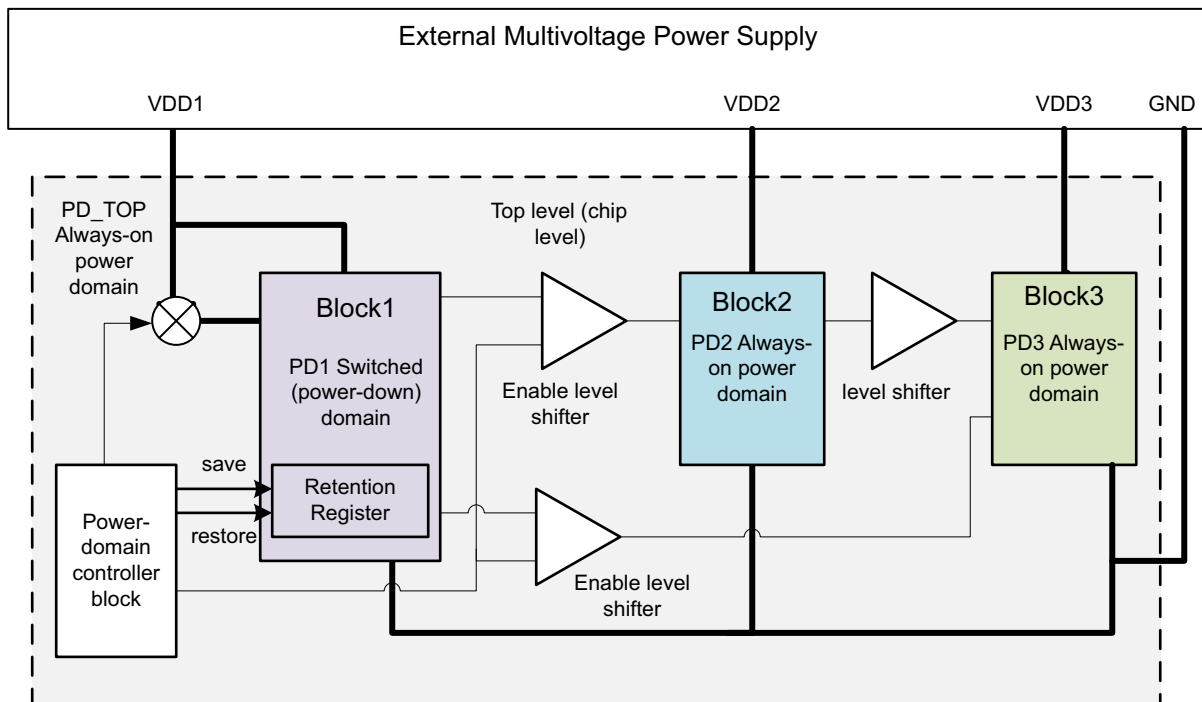
A *level shifter* must be present where a logic signal leaves one power domain and enters another at a substantially different supply voltage. The level shifter converts a signal from the voltage swing of the first domain to that of the second domain.

An *isolation* cell must be present where a logic signal leaves a switchable power domain and enters a different power domain. The level shifter generates a known logic value during shutdown of the domain. If the voltage levels of the two domains are substantially different, the interface cell must be able to perform both level shifting (when the domain is powered

up) and isolation (when the domain is powered down). A cell that can perform both functions is called an *enable level shifter*.

In a power domain that has power switching, any registers that must retain data during shutdown must be implemented as *retention registers*. A retention register has a separate, always-on supply net, sometimes called the backup supply, which keeps the data stable in the retention register while the primary supply of the domain is shut down.

Figure 11-1 Power Intent Specification Example



The power network example shown in [Figure 11-1](#) demonstrates some of the power intent concepts. This chip is designed to operate with three power supplies that are always on (although the UPF syntax also supports externally switchable power supplies), at three different voltage levels. The top-level chip occupies the top-level power domain, PD_TOP. The domain PD_TOP is defined to have four supply ports: VDD1, VDD2, VDD3, and GND. The black squares along the border of the power domain represent the supply ports of that domain. Note that this diagram shows the connections between power domains and is not meant to represent the physical layout of the chip.

In addition to the top-level power domain, PD_TOP, there are three more power domains defined, called PD1, PD2, and PD3, created at the levels of three hierarchical blocks, Block1, Block2, and Block3, respectively. Each block has supply ports (shown as black

squares in the diagram) to allow supply nets to cross from the top level down into the block level.

In this example, PD_TOP, PD2, and PD3 are always-on power domains that operate at different supply voltages, VDD1, VDD2, and VDD3, respectively. PD1 is a power domain that has two supplies: a switchable supply called VDD1g and an always-on supply from VDD1. The always-on power supply maintains the domain's retention registers while VDD1g is powered down.

A power switch shuts off and turns on the power net VDD1g, either by connecting or disconnecting VDD1 and VDD1g. A power-down controller logic block at the top level generates the control signal for the switch. It also generates the save and restore signals for the retention registers in domain PD1 and the control signals for the isolation cells between domain PD1 and the always-on domains PD2 and PD3. These isolation cells generate known signals during times that VDD1g is powered down.

Because domains PD1, PD2, and PD3 operate at different supply voltages, a level shifter must be present where a signal leaves one of these domains and enters another. In the case of the signals leaving PD1 and entering PD2 or PD3, the interface cells must be able to perform both level shifting and isolation functions, because PD1 can be powered down.

For additional background information on UPF terminology and concepts, see the chapter called "Power Intent Specification" in the *Synopsys Multivoltage Flow User Guide*.

UPF Script Example

[Example 11-1](#) shows the UPF script that defines the various concepts supported by UPF.

Example 11-1 UPF Script to Define the Power Intent

```
## CREATE POWER DOMAINS
create_power_domain TOP
create_power_domain PD_ALU -elements {I_ALU} -scope I_ALU
create_power_domain PD_STACK_TOP -elements {I_STACK_TOP} \
-scope I_STACK_TOP
create_power_domain PD_REG_FILE -elements {I_REG_FILE} -scope I_REG_FILE

## SUPPLY NETWORK - PD_ALU
create_supply_net VDD -domain I_ALU/PD_ALU
create_supply_net VSS -domain I_ALU/PD_ALU

create_supply_port VDD -domain I_ALU/PD_ALU
create_supply_port VSS -domain I_ALU/PD_ALU

connect_supply_net I_ALU/VDD -ports {I_ALU/VDD}
connect_supply_net I_ALU/VSS -ports {I_ALU/VSS}

set_domain_supply_net I_ALU/PD_ALU -primary_power_net I_ALU/VDD
-primary_ground_net I_ALU/VSS

## SUPPLY NETWORK - PD_STACK_TOP
create_supply_net VDDT -domain I_STACK_TOP/PD_STACK_TOP
create_supply_net VSS -domain I_STACK_TOP/PD_STACK_TOP

create_supply_port VDDT -domain I_STACK_TOP/PD_STACK_TOP
create_supply_port VSS -domain I_STACK_TOP/PD_STACK_TOP
connect_supply_net I_STACK_TOP/VDDT -ports {I_STACK_TOP/VDDT}
connect_supply_net I_STACK_TOP/VSS -ports {I_STACK_TOP/VSS}

set_domain_supply_net I_STACK_TOP/PD_STACK_TOP
-primary_power_net I_STACK_TOP/VDDT -primary_ground_net I_STACK_TOP/VSS

## SUPPLY NETWORK - PD_REG_FILE
create_supply_net VDDT -domain I_REG_FILE/PD_REG_FILE
create_supply_net VSS -domain I_REG_FILE/PD_REG_FILE

create_supply_port VDDT -domain I_REG_FILE/PD_REG_FILE
create_supply_port VSS -domain I_REG_FILE/PD_REG_FILE

connect_supply_net I_REG_FILE/VDDT -ports {I_REG_FILE/VDDT}
connect_supply_net I_REG_FILE/VSS -ports {I_REG_FILE/VSS}

set_domain_supply_net I_REG_FILE/PD_REG_FILE
-primary_power_net I_REG_FILE/VDDT -primary_ground_net I_REG_FILE/VSS
```

```

## SUPPLY NETWORK - TOP
create_supply_port VDD
create_supply_port VSS
create_supply_port VDDT

create_supply_net VDD -domain TOP
create_supply_net VSS -domain TOP
create_supply_net VDDT -domain TOP

set_domain_supply_net TOP -primary_power_net VDD -primary_ground_net VSS

connect_supply_net VDDT -ports {VDDT I_STACK_TOP/VDDT I_REG_FILE/VDDT}
connect_supply_net VSS
-ports {VSS I_ALU/VSS I_STACK_TOP/VSS I_REG_FILE/VSS}
connect_supply_net VDD -ports {VDD I_ALU/VDD}

## LEVEL-SHIFTER STRATEGY
set_level_shifter ls_alu -domain I_ALU/PD_ALU -applies_to inputs \
-rule both -location self
set_level_shifter ls_stack_top -domain I_STACK_TOP/PD_STACK_TOP \
-applies_to inputs -rule both -location self
set_level_shifter ls_reg_file -domain I_REG_FILE/PD_REG_FILE \
-applies_to inputs -rule both -location self
set_level_shifter ls1_alu -domain I_ALU/PD_ALU -applies_to outputs \
-rule both -location self
set_level_shifter ls1_stack_top -domain I_STACK_TOP/PD_STACK_TOP \
-applies_to outputs -rule both -location parent
set_level_shifter ls1_reg_file -domain I_REG_FILE/PD_REG_FILE \
-applies_to outputs -rule both -location parent

## ISOLATION STRATEGY
set_isolation iso_stack_top -domain I_STACK_TOP/PD_STACK_TOP
-isolation_power_net VDD -isolation_ground_net VSS -clamp_value 1 \
-applies_to outputs -diff_supply_only TRUE
set_isolation iso_reg_file -domain I_REG_FILE/PD_REG_FILE \
-isolation_power_net VDD -isolation_ground_net VSS -clamp_value 1 \
-applies_to outputs -diff_supply_only TRUE

# POWER STATE TABLE
## CREATE PORT STATES
add_port_state VDD -state {TOP 1.08}
add_port_state VDDT -state {BLOCK 0.864} -state {BLOCK_off off}

## OPERATING VOLTAGES
create_pst risc_core_pst -supplies {VDD VDDT}
add_pst_state s0 -pst risc_core_pst -state {TOP BLOCK}
add_pst_state s1 -pst risc_core_pst -state {TOP BLOCK_off}

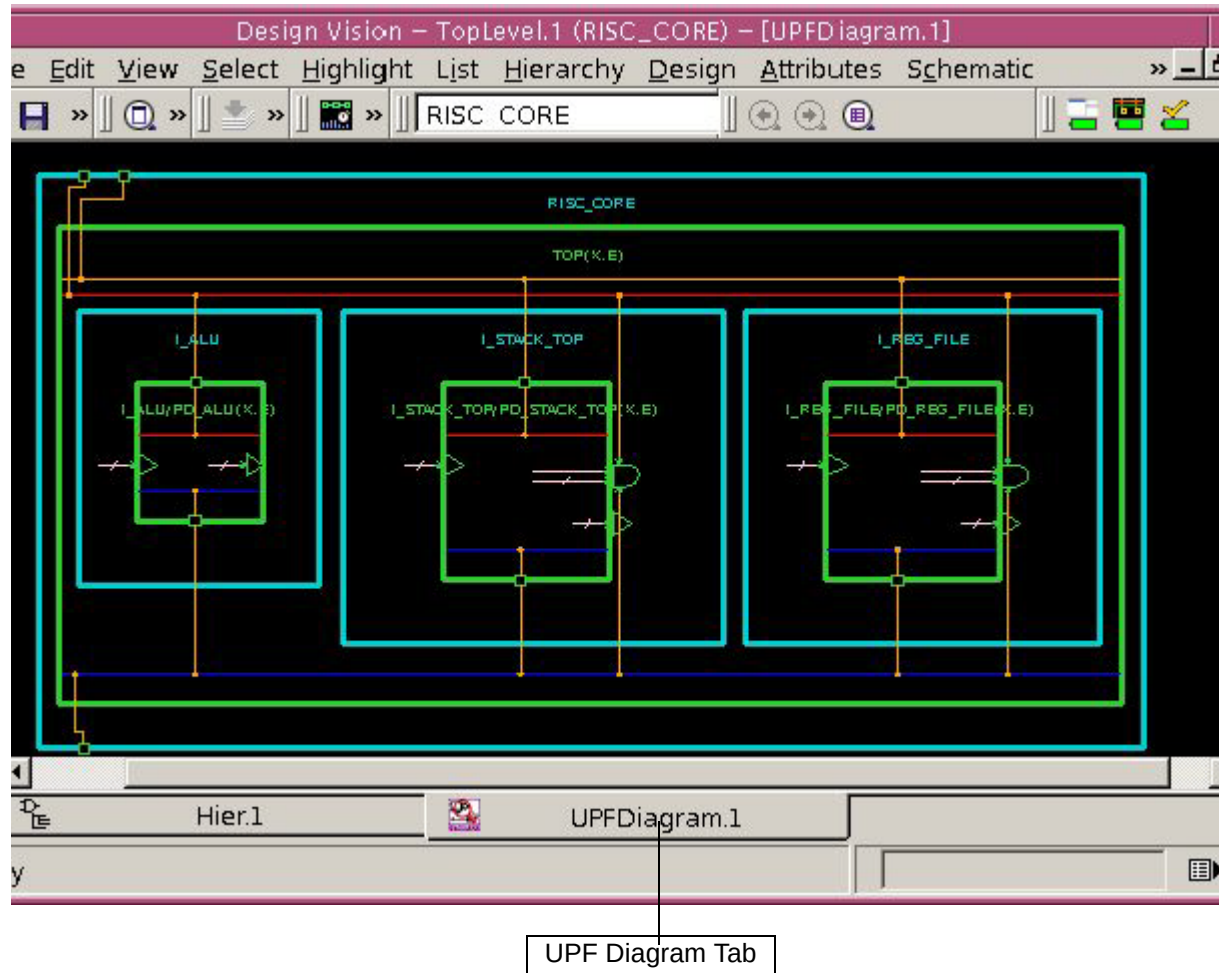
set_port_attributes -elements {I_ALU} -applies_to outputs \
-attribute repeater_power_net I_ALU/VDD \
-attribute repeater_ground_net I_ALU/VSS
set_port_attributes -elements {I_STACK_TOP} -applies_to inputs \
-attribute repeater_power_net VDD -attribute repeater_ground_net VSS

```

```
set_port_attributes -elements {I_REG_FILE} -applies_to inputs \
-attribute repeater_power_net VDD -attribute repeater_ground_net VSS
```

Figure 11-2 shows the UPF Diagram in the Design Vision GUI, for the UPF specification in Example 11-1.

Figure 11-2 UPF Diagram in the GUI for the Specified UPF



Defining Power Intent in UPF

Power Compiler supports the UPF commands to define, review, and examine the power intent specification. Alternatively, you can use the Design Vision GUI to define and examine the power intent specification.

This section discusses in detail, how to use the UPF commands and the GUI to specify the power intent.

Name Spacing Rules for UPF Objects and Attributes

The Power Compiler tool verifies the object names created by the UPF commands, so that the names do not conflict with the names of existing objects in the same logic hierarchy. The tool checks the names of ports, power domains, power state tables, power switches, supply sets and nets, or signal nets. [Table 11-1](#) shows the name spacing rules applied by the tool for the various UPF commands:

Table 11-1 Name Spacing Rules Applied for Various UPF Commands

UPF command names	Name spacing rule
create_power_domain, create_power_switch, create_pst, create_supply_set	Within a logic hierarchy, a power domain cannot have the same name as an existing cell, instance, logic port, supply port, logic net, supply net, power switch, power domain, supply set, or power state table.
create_logic_net, create_supply_net	Within a logic hierarchy, a net cannot have the same name as an existing cell, instance, logic net, supply net, power switch, power domain, supply set, or power state table.
create_logic_port, create_supply_port	Within a logic hierarchy, a port cannot have the same name as an existing cell, instance, logic port, supply port, power switch, power domain, supply set, or power state table.
set_isolation, set_level_shifter, set_retention	The isolation, level-shifter, and retention strategies in a power domain must have unique names.
add_port_state	One or more connected ports cannot have the same port-state names. However, two ports of a mutually connected network can have the same port state (the name and value are same).
add_power_state	A supply set cannot have power states with the same name. However, two ports of a mutually connected network can have the same power state (both name and value are same).
add_pst_state	The power state table cannot have states with the same name as the already existing states.

Defining the Power Intent in the GUI

The Power menu in the GUI allows you to specify, modify, and review your power architecture. It also lets you view the UPF diagram to examine the UPF specification defined in your design.

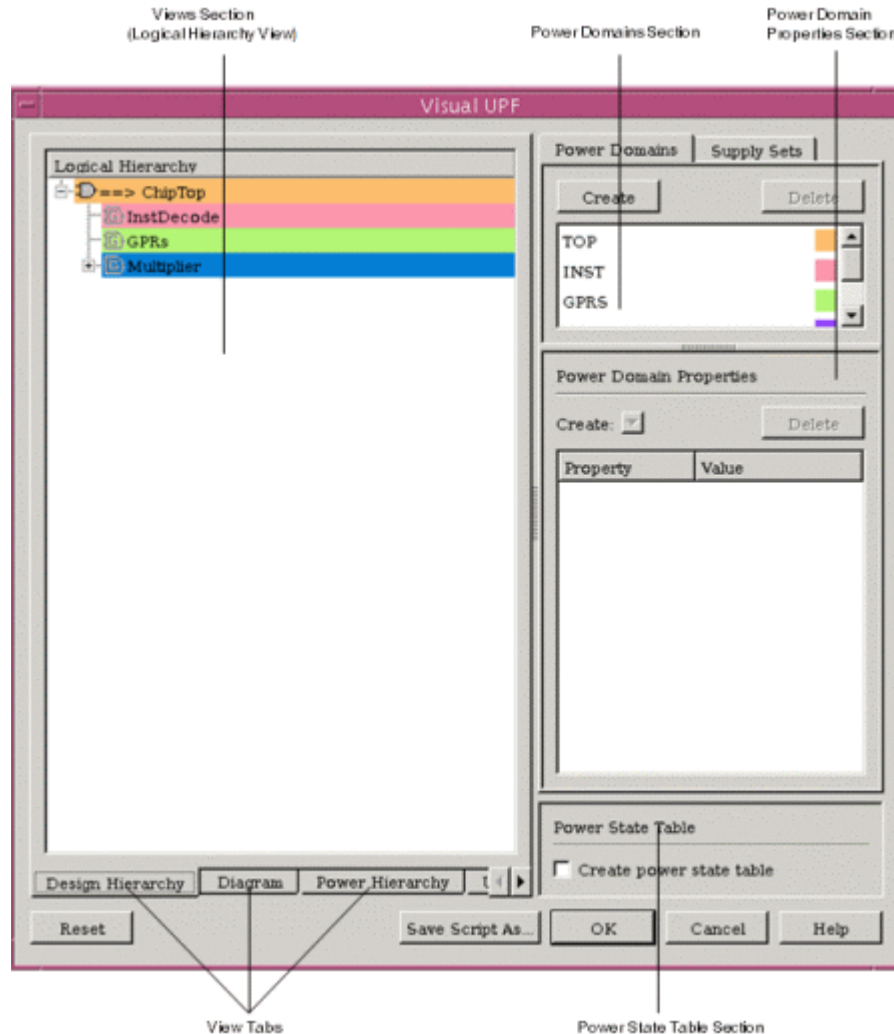
The Visual UPF dialog box in the GUI allows you to define, edit, and review your power intent. You can also generate a UPF script for your power intent.

To open the Visual UPF dialog box,

- Choose Power > Visual UPF

When you open the Visual UPF dialog box, it appears as shown in [Figure 11-3](#).

Figure 11-3 Logic Hierarchy View of the Visual UPF



If you have not yet defined the power intent for your design, use the Power Domains and Power Domain Properties sections to create the power domains and various other components, such as power-switches, level-shifters and so on. For the first power domain that you create, the tool assigns the name TOP by default.

If you have already defined the power intent for your design, the Visual UPF displays the details of your power specification. Using the Power Domains and Power Domain Properties sections, you can edit the power definitions: add new components, redefine the association of the hierarchical cells with the power domains, delete a power domain, and so on.

For more details about how to review the UPF intent in the GUI, see [Reviewing the Power Intent Using the Design Vision GUI](#).

UPF Diagram View

The UPF diagram view displays the UPF power intent as it is defined in the design database. When you change the database, for example, by entering a UPF command, the tool reflects the updates in the UPF diagram immediately. You can view the UPF diagram at any point in the design flow.

To open the UPF diagram view

- Choose Power > UPF Diagram > New UPF Diagram View.

When the UPF diagram view appears, Design Vision displays a tab at the bottom of the workspace area, as shown in [Figure 11-3](#). You can use this tab to return to the UPF diagram view after working with other views.

The UPF diagram view represents each power object with a unique symbol (for more information about these symbols, see the “UPF Diagram Symbols and Standards” topic in Design Vision Help). It uses default colors to differentiate the various types of power objects. You can customize the diagram by using the View Settings panel, to change object colors or apply a color theme.

For more information, see the “Changing UPF Diagram Display Properties” topic in Design Vision Help.

Creating Power Domains

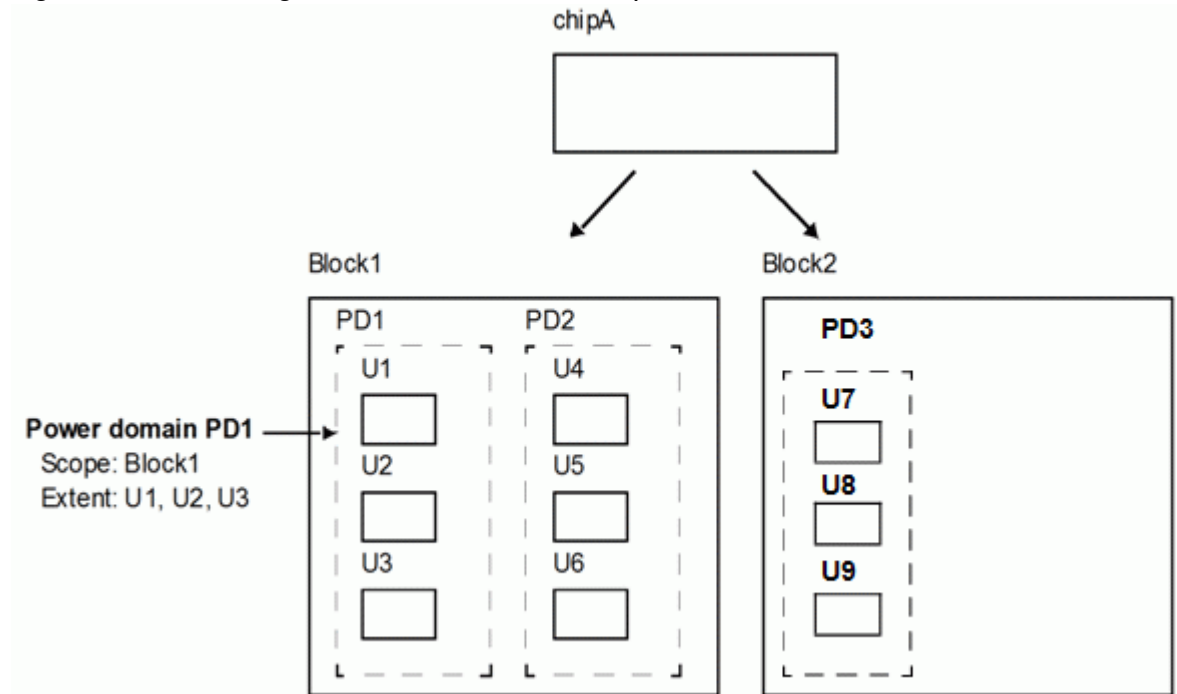
To create a power domain, use the `create_power_domain` command.

The `-elements` option specifies the list of hierarchical, macro and pad cells that are added to the extent of the power domain. If the required scope is at a lower level than the current scope, use the `-scope` option to specify the name of the instance where the power domain is to be defined.

The UPF standard requires a simple name for the *domain_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to `true`.

Figure 11-4 shows the usage of the `create_power_domain` command.

Figure 11-4 Defining a Power Domain and Scope



To create the PD1 and PD2 power domains, use the following commands:

```
create_power_domain -elements {U1 U2 U3} -scope Block1 PD1
create_power_domain -elements {U4 U5 U6} -scope Block1 PD2
```

Alternatively, you can use the `set_scope` command to first set to the desired scope and then to create the power domain, as mentioned in the following example:

```
set_scope Block1
create_power_domain -elements {U1 U2 U3} PD1
create_power_domain -elements {U4 U5 U6} PD2
```

You can use the `-include_scope` option to include all the elements in the specified scope to share the supply of the power domain.

```
create_power_domain -elements {U7 U8} -include_scope Block2 PD3
```

In this case, the element U9 shares the supply of power domain PD3, though U9 is not explicitly specified as part of the power domain PD3.

To add new elements to any hierarchy, except those that are already specified as an element of another power domain, use the `-update` option. You cannot re-specify an existing element. You can only add new elements.

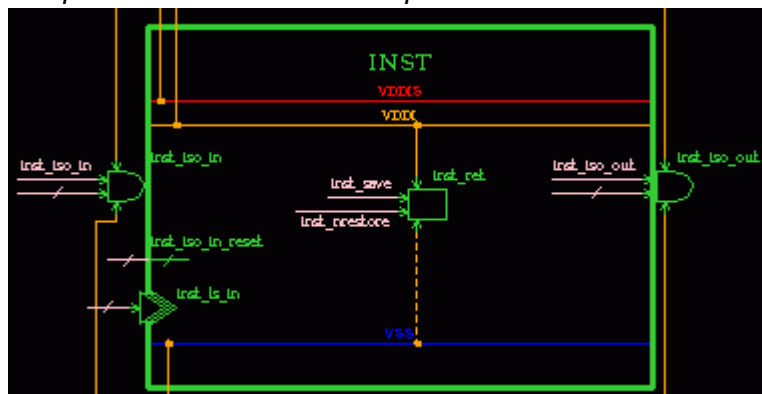
Representation of Power Domain in the UPF Diagram View

The UPF diagram view displays all power domains that are defined in the current design and its subdesigns. The power domains are organized hierarchically, such that each power domain is located inside its parent power domain.

A power domain is represented by a green colored rectangular bounding box. The name of the power domain is displayed inside the bounding box. [Figure 11-5](#) shows the INST power domain and all the UPF objects contained in the power domain.

The size of the power domain symbol varies according to the number and size of the objects that reside within the power domain. The symbol is big enough to display all the UPF objects that are contained in it.

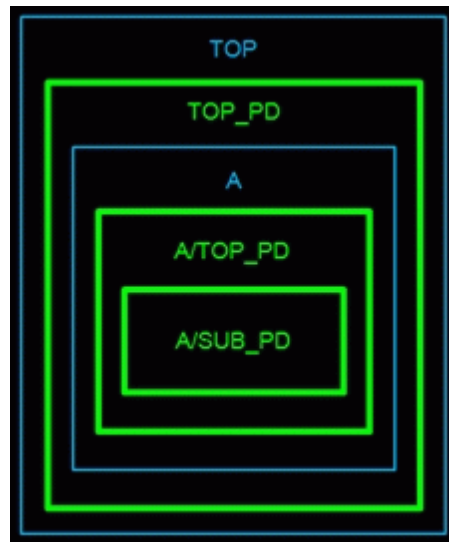
Figure 11-5 An Example of a Power Domain Representation in the UPF Diagram



Scope

In the UPF diagram view, scope is represented by a blue colored rectangular bounding box. The scope appears within the hierarchy of the power domains. The bounding box of the scope surrounds the top-most child domain in the scope. [Figure 11-6](#) shows an example of how power domains and scopes appear within the UPF diagram.

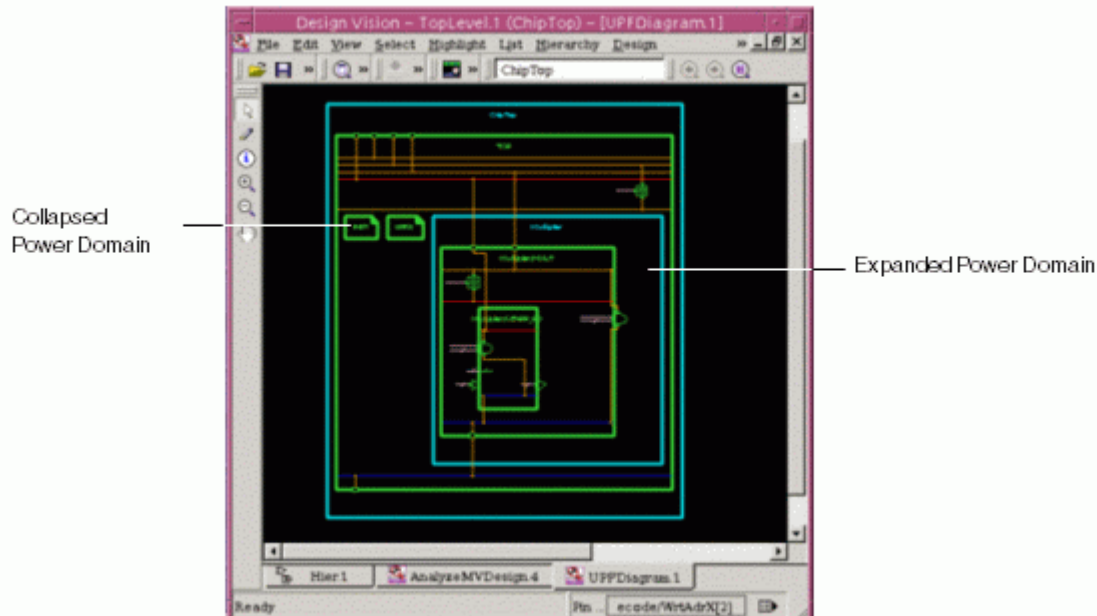
Figure 11-6 Representation of Power Domains and Scopes in the UPF Diagram



Expanding and Collapsing Power Domains in the GUI

In the UPF diagram view, you can collapse or expand a selected power domain or scope. This is useful when you have large designs with several power domains. When you open the UPF diagram view, by default the power domains are expanded, as shown in [Figure 11-2](#). When you collapse a power domain, all its internal details disappear from the view, and only its name is displayed, as shown in [Figure 11-7](#). When you expand a power domain, all its internal details are displayed in the view.

Figure 11-7 UPF Diagram With Collapsed Power Domains



You can use either of the following methods to expand or collapse a power domain.

After selecting one or more power domains that you want to expand,

- Choose Power > UPF Diagram > Expand Selected Domains.
- Right-click the diagram and choose Expand Selected Domains.

After selecting one or more power domains that you want to collapse,

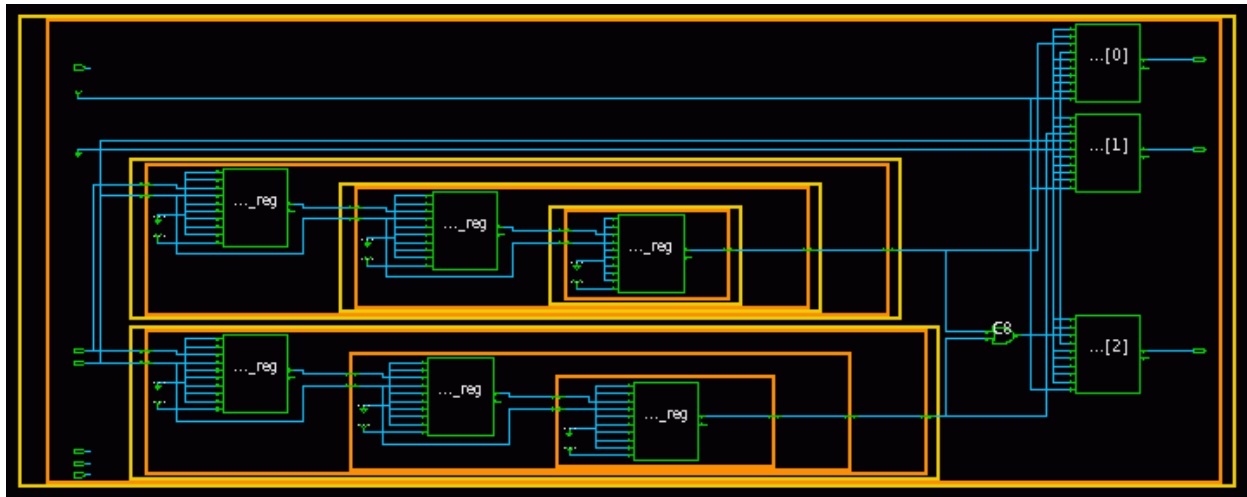
- Choose Power > UPF Diagram > Collapse Selected Domains.
- Right-click the diagram and choose Collapse Selected Domains.

Viewing Hierarchical Cell and Power Domain Boundaries

By default, the schematic view displays timing paths and design logic in a flat, single-sheet schematic that can span multiple hierarchy levels. Hierarchy crossings are represented by diamond shapes and indicate where the nets traverse a level of hierarchy.

You can improve your view of the hierarchical structures in the design by arranging the schematic to display objects hierarchically. Rectangular boundaries appear around objects that share the same hierarchical parent block. Hierarchical cell boundaries appear orange and power domain boundaries appear yellow as shown in [Figure 11-8](#).

Figure 11-8 Hierarchical and Power Domain Boundaries



To display or hide hierarchical boundaries in the active schematic view,

- Choose Schematic > Show Logic/Power Hierarchy.

A check mark beside the command on the Schematic menu indicates that the boundaries are visible.

You can color the objects in a schematic based on the hierarchical power relationships of the design.

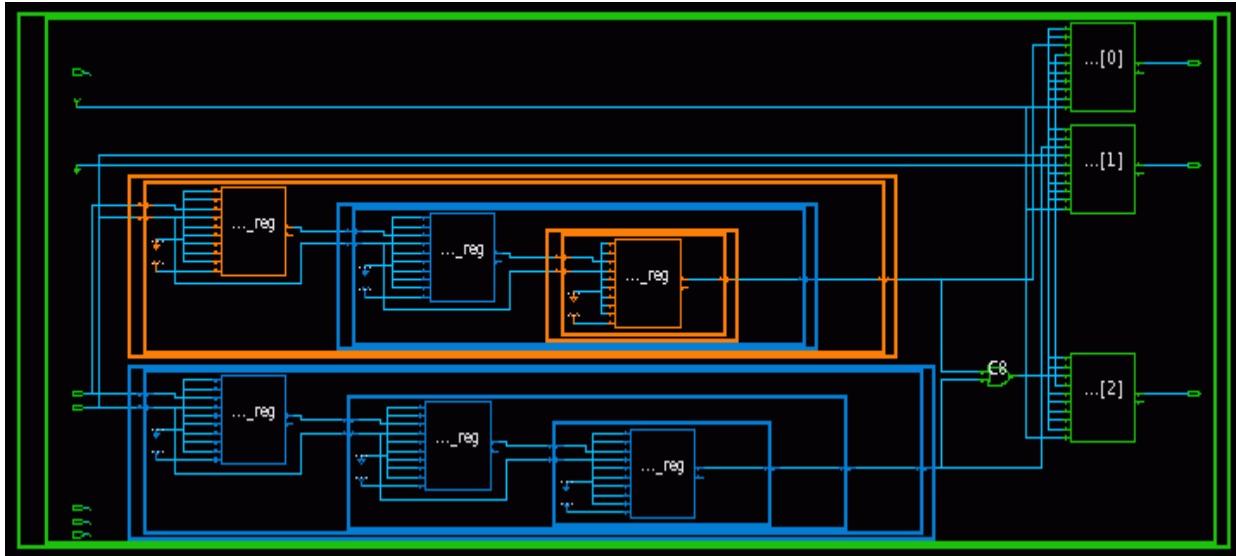
To display or hide boundary coloring based on their power domains,

- Choose Schematic > Color by Power Hierarchy.

A check mark beside the command on the Schematic menu indicates that the boundary coloring is visible. The tool displays the objects for each power domain with a unique color.

In [Figure 11-9](#), PD_TOP power domain and its elements appear green. PDA power domain and its elements appear orange and PDB power domain and its elements appear blue.

Figure 11-9 Hierarchical and Power Domain Boundaries Colored by Power Hierarchy



For more information, see the “Viewing Cell and Power Domain Hierarchies” topic in Design Vision Help.

Creating Supply Ports

To create the power supply and ground ports, use the `create_supply_port` command.

The name of the supply port should be a simple (non-hierarchical) name and unique at the level of hierarchy it is defined. Unless the `-domain` option is specified, the port is created in the current scope or level of hierarchy and all power domains in the current scope can use the created port. By default, the direction of the port is in (or input port).

The UPF standard requires a simple name for the `port_name` argument of the command. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to `true`.

The following example shows how to create the ports shown in [Figure 11-1](#).

To create the supply ports VDD1, VDD2 and VDD3 and GND at the top level of design hierarchy or power domain PD_TOP use the command as follows:

```
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VDD3
create_supply_port GND
```


To create the supply ports VDD1, VDD1g and GND in the power domain PD1, use the `create_supply_port` command as follows:

```
create_supply_port VDD1 -domain PD1
create_supply_port VDD1g -domain PD1
create_supply_port GND -domain PD1
```

To create the supply ports VDD2 and GND in the power domain PD2 and VDD3 and GND in power domain PD3, use the `create_supply_port` command as follows:

```
create_supply_port VDD2 -domain PD2
create_supply_port GND -domain PD2
create_supply_port VDD3 -domain PD3
create_supply_port GND -domain PD3
```

Note:

Connectivity is not defined when the supply port is created. To define connectivity use the `connect_supply_net` command.

Adding Port State Information to Supply Ports

The `add_port_state` command adds state information to a supply port. This command specifies the name of the supply port and the possible states of the port. The first state specified is the default state of the supply port. The port name can be a hierarchical name. Each state is specified as a state name and the voltage level for that state. The voltage level can be specified as a single nominal value, set of three values (minimum, nominal, and maximum), or 0.0, or the keyword `off` to indicate the off state. The state names are also used to define all possible operating states in the Power State Table.

Note that supply states specified at different supply ports are shared within a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

[Example 11-2](#) shows the definition of states for the power nets:

Example 11-2 Defining the States of the Power Nets

```
dc_shell> add_port_state header_sw/VDD -state {HV 0.99} -state {LV 0.792}
          -state {OFF off}
```

[Example 11-3](#) shows the definition of states for the ground nets:

Example 11-3 Defining the States of the Ground Nets

```
dc_shell> add_port_state footer_sw/VSS -state {LV 0.0} -state {OFF off}
```

[Example 11-4](#) has the HV1_1 and HV2_1 states with the same voltage value, 1.2, on the VDD1 supply port. The duplicate port states are useful in hierarchical flow, where the top level and block-level ports have different state names but the same voltage value.

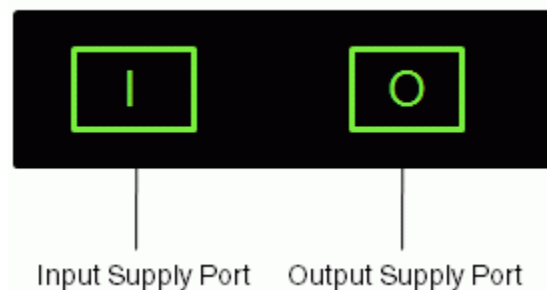
Example 11-4 Defining Duplicate Port States

```
dc_shell> add_port_state VDD1 -state {HV1_1 1.2} -state {HV2_1 1.2}
```

Representation of Supply Ports in the UPF Diagram View

In the UPF diagram view, a supply port is represented by a bounding box. A letter in the bounding box indicates the direction of the port, as shown in [Figure 11-10](#).

Figure 11-10 Representation of Power Supply Port in the UPF Diagram



The UPF diagram displays all the supply ports in the current design and its subdesigns. It also shows the connectivity of the supply ports with the supply nets, their locations, and the power domains where they belong.

Supply ports are located on the border of the power domain where they belong. They are located at the top or at the bottom boundary of the power domain, depending on the supply net connected to the supply ports. In addition, input ports are located on the left side, and the output ports are located on the right side.

Creating Supply Nets

A supply net connects supply ports or supply pins. To create a supply net, use the `create_supply_net` command.

The supply net is created in the same scope or logic hierarchy as the specified power domain. When you use the `-reuse` option, the specified supply net is not created; instead an existing supply net with the specified name is reused.

```
create_supply_net GND_NET -domain PD1
create_supply_net GND_NET -domain PD2 -reuse
```

When a supply net is created, it is not considered a primary power supply or ground net. To make a specific power supply or ground net of a power domain, the primary supply or ground net, use the `set_domain_supply_net` command.

The UPF standard requires a simple name for the `net_name` argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to `true`.

All supply nets including the ground, must be assigned an operating voltage value. If any supply net does not have an assigned operating voltage, Power Compiler issues UPF-057 error message during the execution of the `compile_ultra` command. Before compiling the design, use the `check_mv_design -power_nets` command to ensure that operating voltages are defined for all the supply nets. For more details, see [Examining and Debugging UPF Specifications](#).

The operating voltage that you have already set cannot be removed. However, you can override the existing settings by using the `set_voltage` command again.

Specifying Primary Supply Nets for a Power Domain

To define the primary power supply net and primary ground net for a power domain, use the `set_domain_supply_net` command.

Every power domain must have one primary power and one ground connection. When a supply net is created it is not a primary supply net. You must use the `set_domain_supply_net` command to designate the specific supply net as the primary supply net for the power domain. All cells in a power domain are assumed to be connected to the primary power and ground net of the power domain. If the power or ground pins of a cell in a power domain, is not explicitly connected to any supply net, the power or ground pin of the cell is assumed to be connected to the primary power or ground net of the power domain to which the cell belongs.

The following example shows the commands to specify VDD and GND nets as the primary power and ground net, respectively, of the PD_TOP power domain.

```
dc_shell> set_domain_supply_net -primary_power_net VDD \  
-primary_ground_net GND PD_TOP
```

Note:

If you use supply sets to define the primary supply and ground, the supply nets that you specify must belong to the same supply set. Otherwise Power Compiler issues an error message. For more details see, [Specifying Supply Sets](#).

Representing Supply Nets in the UPF Diagram View

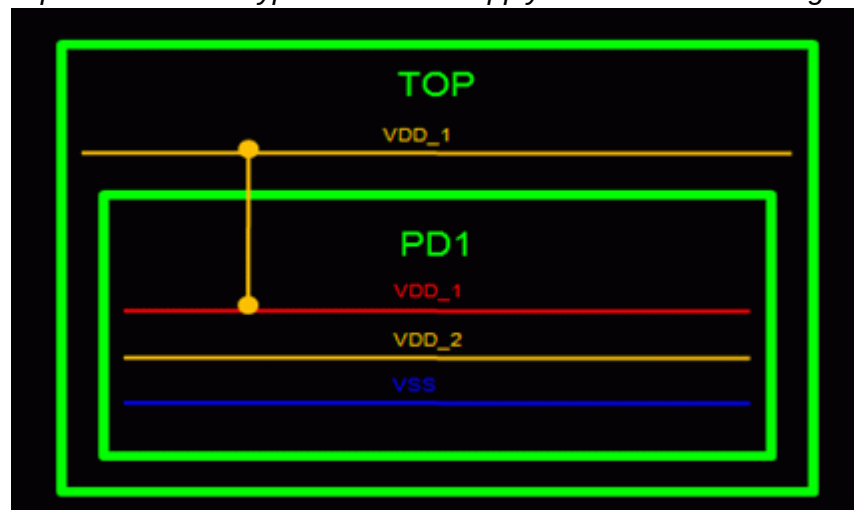
In the UPF diagram view, a supply net is represented by a line or a line segment. Different colors are used to differentiate the type of the net, as shown in [Table 11-2](#).

Table 11-2 Colors Used to Represent Types of Net Segments

Color	Net segment
Red	Primary power net
Blue	Primary ground net
Yellow	All other net segments

As shown in [Figure 11-11](#), the UPF diagram view displays all the supply nets in the current design and the current design's subdesigns, and their supply net connections.

Figure 11-11 Representation of Types of Power Supply Nets in the UPF Diagram



The location of the supply nets in the diagram is based on the location of the power domains where they belong and also on the type of the supply net. Each power domain that a supply net belongs to contains a line segment indicating the supply net.

Horizontal segments represent supply nets inside the power domain. Vertical segments represent nets that are reused in multiple power domains and that are connected to another object, such as a supply port or a power switch.

Power supplies extend down from the top of the power domain, and ground nets extend up from the bottom of the power domain.

In [Figure 11-11](#), the VDD_1 net is the primary supply net of PD1 power domain. However, it is not the primary supply net of the power domain TOP. Similarly, VSS is the primary ground net of power domain PD1.

Connecting Supply Nets

The `connect_supply_net` command connects the supply net to the specified supply ports or pins. The connection can be within the same level of hierarchy or to ports or pins down the hierarchy.

You can also use the `connect_supply_net` command to connect to the internal PG pins of macro cells containing fine-grained switches. For more information about macro cells with fine-grained switches, see [Macro Cells with Fine-Grained Switches](#).

The UPF standard requires a simple name for the *supply_net_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to `true`.

The following example shows the use of the `connect_supply_net` command to connect supply nets to various supply ports at different levels of hierarchy or power domains.

```
connect_supply_net GND_NET -ports GND
connect_supply_net GND_NET -ports {B1/GND B2/GND B3/GND} GND
```

You can also use the function of a supply set with the `connect_supply_net` command, as shown in the following example:

```
create_supply_set ss
connect_supply_net ss.ground -ports {B1/GND}
```

Use the `create_supply_net -resolve parallel` command when

- A supply net connects to the internal PG pins of more than one macro cell with a fine-grained switch.
- A supply net is associated with a supply set group that has multiple drivers at the scope of the supply net; the net should be created using the `create_supply_net -resolve parallel` option. For more information about associating supply sets, see [Associating Supply Sets With Supply Set Handles](#).

Note:

The `connect_supply_net` command ignores connections to the pins of physical-only cells.

Converting the PG Information in RTL to UPF

When your RTL design contains PG nets and pin connections to macro, I/O, and power management cells, Power Compiler can convert these details into UPF constraints. Follow these steps to convert the PG information in RTL into UPF constraints and use these constraints during synthesis:

1. Set the `dc_allow_rtl_pg` variable to `true`.
2. Read the RTL design.
3. Link the design.
4. Load the UPF file.
5. Run the `convert_pg` command.
6. Run the `compile_ultra` command to optimize the design.

When the RTL design has PG connection details and the power constraints are specified in UPF, you must specify the `convert_pg` command before you run the following commands:

- `compile_ultra`
- `insert_mv_cells`
- `insert_dft`
- `dft_drc`
- `analyze_mv_design`
- `check_mv_design`
- `save_upf`

For more details, see the `dc_allow_rtl_pg` variable man page and `convert_pg` command man page.

Specifying Supply Sets

A supply set is an abstract collection of supply nets consisting of two supply functions: power and ground. A supply set is domain-independent, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an

abstraction of the supply nets and supply ports needed to power a design. Before such a design can physically implemented (placed and routed), its supply sets must be refined, or associated with actual supply nets.

A supply set consists of the following two functions:

- Power
- Ground

You can access the functions of the supply set by using the name of the supply set and the name of the function. To access the power function of the supply set *SS*, specify *SS.power*. To access the ground function of the supply set *SS*, specify *SS.ground*.

Creating Supply Sets

To create a supply set, use the `create_supply_set` command. The supply set is created in the current logic hierarchy or the scope.

The UPF standard requires a simple name for the *supply_set_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to `true`.

The following example shows how to create a supply set and associate it with the primary power supply of a power domain:

```
create_supply_set primary_supply_set
create_power_domain PD_TOP
set_domain_supply_net PD_TOP \
  -primary_power_net primary_supply_set.power \
  -primary_ground_net primary_supply_set.ground
```

Note:

When you specify a supply set as the primary power and ground supply of the power domain, both the primary and the ground supply must belong to the same supply set.

In the UPF Diagram view, a supply set does not appear visually in the diagram. Only the supply nets of a supply set appear in the diagram. Supply nets of a supply set and domain-independent supply nets are implicitly available anywhere from their scope downward in the design.

Creating Supply Set Handles

When you create a power domain, the following supply set handles are created by default:

- `primary`
- `default_isolation`
- `default_retention`

In addition to these predefined supply set handles, you can define supply set handles by using the `-supply` option of the `create_power_domain` command. To associate multiple supply sets with a power domain, use the `-supply` option multiple times.

Supply set handles are created at the scope of the power domain and are available for use in the power domains that are at the same or lower scope than the power domains where they are created. Use the following naming convention to refer to a supply set handle: *power_domain_name.supply_set_handle*. When a power domain is deleted, its supply set handles are also deleted.

To disable the creation of supply set handles while creating the power domain, set the `upf_create_implicit_supply_sets` variable to `false` before you load the UPF file.

Note:

After loading the UPF file, the `upf_create_implicit_supply_sets` variable becomes a read-only variable and you can no longer change its value.

You can also specify the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command to restrict the availability of the supplies in the power domain. For more information about using the `extra_supplies_#` keyword, see [Restricting Supply Sets Available to a Power Domain](#).

The following example shows how to create a power domain and associate a supply set with the power domain:

```
# Create the supply sets
create_supply_set primary_supply_set

# Create power domain and associate it with the supply set
create_power_domain PD1 -supply {primary primary_supply_set}
```

Restricting Supply Sets Available to a Power Domain

Supply sets are domain-independent and can only be updated with domain-independent nets. To restrict the supply sets available to a power domain, use the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
dc_shell> create_power_domain SUB_DOMAIN \  
          -supply {extra_supplies_1 supply_set1} \  
          -supply {extra_supplies_2 supply_set2} -elements mid1/PD_MID
```

Alternatively, if you do not want the power domain to use extra supply nets other than those that are already defined in other strategies, specify the `extra_supplies ""` keyword (without the index) with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
dc_shell> create_power_domain PD_MID -scope mid1 \  
          -supply {extra_supplies ""}
```

It is an error to use both the `extra_supplies_#` and `extra_supplies ""` keywords simultaneously.

By default, a power domain can use domain-independent supply nets and supply nets defined in the power domain. However, when you define supply sets with the `extra_supplies_#` keyword, the power domain is restricted to use

- The primary supply of the power domain
- The supplies listed as `extra_supplies_#`
- The supplies specified by the isolation strategy of the power domain
- The supplies specified by the retention strategy of the power domain
- The supplies defined or reused as domain-dependent supplies in the power domain

Refining Supply Sets

To redefine the functions of a supply set, use the `-update` option of the `create_supply_set` command. You must use the `-update` and the `-function` options together, to associate the function names with the supply nets or ports.

The following example shows how you use the `-update` option to associate supply nets to the functions of the supply set:

```
create_power_domain PD_TOP
create_supply_net TOP_VDD
create_supply_net TOP_VSS
create_supply_set supply_set \
  -function {power TOP_VDD} \
  -function {ground TOP_VSS} \
  -update
```

The following rules apply, when you update a supply set with a supply net:

- **Voltage rule**
 The voltage of the supply set handle must match with the voltage of the supply net with which the supply set is updated.
 If voltage is not specified for the supply net, then after the update, the voltage on the supply set handle is inferred as the voltage of the supply net.
- **Function rule**
 The supply set function must match with the function of the supply net with which the supply set is updated.
 Power Compiler issues an error message when,
 - The ground handle of a supply set is used to update power handle of another supply set and vice versa.
 - The supply net updated with the ground handle of a supply set is connected to a power supply port or pin of a power object, such as a power domain, and vice versa.
- **Scope rule**
 The scope of supply set must match with the scope of the explicit supply net with which the supply set is updated.
- **Availability rule**
 The explicit supply net with which the supply set is updated, must be domain-independent.
- **Connection rule**
 The explicit supply net with which the supply set is updated, should not be connected to a driver port when the supply set handle is connected to a driver port unless a resolution function is defined for the explicit supply net.
- **Conflicting supply state names rule**
 A supply set handle cannot be updated with a supply net or supply set if their power states are not identical.

- Valid power-state-table rule

When a number of supply sets are updated to the same supply net, only one supply set can be present in the power state table.

Associating Supply Sets With Supply Set Handles

A predefined supply set can be referenced through a supply set handle. Additional supply net functions can also be defined for a predefined supply set. A supply set handle is created at the scope of the power domain, and is available for use in all domains at and above the scope where it is created. The tool issues an error message if you use the supply set handle outside the scope where it is created.

When a supply set handle is associated with a supply set, the tool considers the two supply sets to be connected, and their functions resolve to the same supply nets. To associate a supply set handle to another supply set, use the `associate_supply_set` command.

The following command

```
create_supply_set PD1.primary -function {power VDD} \  
    -function {ground VSS}
```

is equivalent to the following set of commands

```
create_supply_set my_sset -function {power VDD} \  
    -function {ground VSS}  
associate_supply_set my_sset -handle PD1.primary
```

Rules for Associating Supply Sets

The following rules apply, when you associate a supply set with a supply set handle.

- Associating a supply set handle to a supply set can be done only one time.
- Associating a supply set handle to a supply set should not cause circular associations.
- User-defined supply sets cannot be specified with the `-handle` option of the `associate_supply_set` command.
- The supply set handle specified with the `-handle` option of the `associate_supply_set` command must be at the same or below the scope of the specified supply set.
- While associating a supply set handle to supply set, the supply set must be available in the power domain where the supply set handle is available.

Defining Power States for the Components of a Supply Set

Power states are attributes of a supply set. The supply nets of a supply set can be at different power states at different times. Using the `add_power_state` command, you can define one power state for all those supply nets of the supply set that always occur together. For each power state of the supply set, you must use one `add_power_state` command. By default, the undefined power states are considered illegal states.

Use the `-state` option to specify the name of the power state of the supply set.

Use the `-supply_expr` option to specify the power state and the voltage value for the various supply net components of the supply set as shown in the following example:

```
add_power_state supply_set_name \
  -state state_name \
  -supply_expr { supply_net_function == \
                  {legal_state, [voltage_1, \
                                [voltage_2, \
                                [voltage_3]]]}}
```

The expression specified with the `-supply_expr` option is used to determine the legal states of the supply nets of the supply set during the synthesis of the design. You can specify only the following allowed states:

- FULL_ON
- OFF

For each state of the supply net component you can specify up to three voltage values using floating point numbers. You can defer specifying the voltage value and specify it later using the `add_power_state -update` option.

The voltage values that you specify with a power state are interpreted by the tool as follows:

- When you specify no voltage value, the tool assumes that the value will be specified at a later time with the `add_power_state -update` option.
- When you specify a single voltage value, this value is considered the nominal voltage of the associated state.
- When you specify two voltage values, the first value is considered the minimum voltage and the second as the maximum voltage. The average of the two values is considered as the nominal voltage of the power state.
- When you specify three voltage values, the first value is considered as the minimum voltage, the second as the nominal and the third as the maximum voltage of the power state.

Note:

The tool issues an error message if the second value is less than the first and the third value is less than the second.

To incrementally specify a power state definition, you can issue multiple `add_power_state -supply_expr` commands. To do this, use the `-update` option on subsequent commands to incrementally update the power state definition. For example,

```
add_power_state SS1 -state ON {-supply_expr {power == {FULL_ON 0.8}}}
add_power_state SS1 -state ON \
{-supply_expr {ground == {FULL_ON 0.0}}} -update
```

When you define a power state with two `add_power_state -supply_expr` commands, the supply expression becomes a logical AND of the two definitions. Therefore, the previous two `add_power_state` commands for the ON state are equivalent to the following:

```
add_power_state SS1 -state ON \
{-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
```

The `add_power_state` command supports the `-logic_expr` option which is parsed but ignored by Power Compiler.

The UPF standard requires a simple name for the *object_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to true.

The following example shows the use of the `add_power_state` command to define the power states HVp and HVg for the components of the supply set, `PD1_primary_supply_set`:

```
dc_shell> add_power_state PD1_primary_supply_set -state HVp \
{-supply_expr {power == {FULL_ON, 1.08, 2.05, 3.0}}}
```

```
dc_shell> add_power_state PD1_primary_supply_set -state HVg \
{-supply_expr {ground == {FULL_ON, 0.0}}}
```

For more information about the `add_power_state` command, see the *Synopsys Multivoltage Flow User Guide*.

Correlated Grouping of Supply Voltage Triplets

If the voltage variation for each supply is correlated with, not independent of, the other supplies, you can define the supplies as correlated, so that the tool considers only the minimum with minimum voltages, only nominal with nominal voltages, and only maximum with maximum voltages, without mixing between minimum, nominal, and maximum. This method of analysis is called correlated grouping of voltage triplets.

Power Compiler supports correlated grouping of the minimum, nominal, and maximum voltages specified as triplets in the `add_port_state` and `add_power_state` commands. To define one or more groups of correlated supply nets, use the `correlated_supply_group`

attribute with the `set_design_attributes` command. For example, the following command groups VDD1 and VDD2 supply nets into a correlated supply group and sets the supply voltages as correlated triplets:

```
set_design_attributes -elements {.} \  
-attribute correlated_supply_group "{VDD1 VDD2}"
```

You can define the port state and power state table as follows:

```
add_port_state VDD1 -state {HV 0.9 1.0 1.1}  
add_port_state VDD2 -state {HV 1.0 1.1 1.1}  
add_port_state VSS -state {ON 0.0}  
  
create_pst PST -supplies {VDD1 VDD2 VSS}  
add_pst_state HV_STATE -pst PST -state {HV HV ON}
```

The tool analyzes the design behavior with correlated VDD1 and VDD2 supplies, without mixing between minimum, nominal, and maximum voltages.

For more information about using the `set_design_attributes` command, see [“Setting Attributes on Hierarchical Cells” on page 11-59](#).

Always-On Logic

Generally, multivoltage designs have power domains that are shut down and powered up during the operation of the chip while other power domains remain powered up. The control nets that connect cells in an always-on power domain to cells within the shut-down power domain must remain on during shutdown. These paths are referred to as always-on paths.

Marking Library Cells as Always-On

Power Compiler performs always-on optimization, only when the target library contains always-on inverters and always-on buffers. To use a specific library cell in the optimization of always-on paths within the shut-down power domains, mark the cell with the `always_on` attribute. The tool uses only the always-on cells to optimize the always-on paths within the shut-down power domains. The cells that are not marked as always-on are used outside the shut-down power domains.

Note:

When you set the `always-on` attribute on a library cell, the tool does not use the library cell for optimization of other types of paths. To use a library cell in both always-on paths and shut-down paths, set the `always-on` attribute only on the instances of the library cell that are present in the shut-down power domains.

Marking Pass-Gate Library Pins

Power Compiler can prevent connecting always-on cells to cells with pass-gate inputs. An always-on cell should not drive a gate that has pass transistors at the inputs (pass-gate). Pass-gate input cells should be driven by a standard cell in a shut-down power domain. Therefore, if your library contains any of these cells, you must mark them as pass-gates in each session.

The following example shows how to mark pin A of the MUX1 cell, with the `pass_gate` attribute.

```
dc_shell> set_attribute [get_lib_pins Lib_name/MUX1/A] pass_gate true
```

Always-On Optimization

The Power Compiler tool constrains, marks, and optimizes always-on nets, including feedthrough nets. The tool considers the `mv_make_primary_supply_available_for_always_on` variable when selecting which power supply to use for inserted buffers. By default, the variable is `true` and the tool uses the domain's primary supply, the related supply net of the load, or the driver pin as the supply net for the inserted buffers. When the variable is set to `true`, the tool inserts regular buffers instead of always-on buffers on feedthrough nets when the primary power supply can be used to power the buffers without introducing electrical violations. To give preference to load and driver supplies, do the following:

```
set_app_var mv_make_primary_supply_available_for_always_on false
```

The tool also ensures that no additional isolation or level-shifting violations are introduced by the automatic always-on synthesis. If the isolation strategy is specified with the `-source` and `-sink` options, the tool preserves the original source and sink relationship on these paths.

To determine the supply nets used for the buffers and inverters inserted during always-on synthesis when the variable is set to `true`, the tool applies the following rules, in the specified order:

1. Highest precedence is given to the domain's primary supply.
2. For a load net, when the related supply net of the load is in the same power domain as the net, the related supply net of the load is used.
3. For a driver net, when the related supply net of the driver is in the same power domain as the net, the related supply net of the driver is used.
4. For a feedthrough net with multiple choices of nets, highest precedence is given to the domain's primary supply. The related supply net of the load takes precedence over the related supply net of the driver.

When the `mv_make_primary_supply_available_for_always_on` variable is set to `false`, the tool does not use the domain's primary supply, and instead uses the related supply net of the load or the driver pin as the supply net for the inserted buffers. Otherwise, the determination of the supplies for the inserted buffers is the same. With the variable set to `false`, the tool gives preference to the load and driver supplies and as a result, more always-on buffers might appear in the netlist.

Regardless of how the `mv_make_primary_supply_available_for_always_on` variable is set, the tool marks the selected nets based on the following rules:

- When the related supply net is in the same power domain as the net and it is not the primary power net of the power domain, the tool marks the net as `always_on`.
- When the related supply net is not in the same power domain as the net, the tool marks the net as `dont_touch`.
- When the related supply net is in the same power domain as the net, and it is the primary power net of the power domain, the tool inserts a regular buffer or inverter, and the net is not marked.

Voltage-Aware Always-On Synthesis

When running the Design Compiler tool in topographical mode, you can enable voltage-aware always-on synthesis on certain physical feedthrough paths. This enables buffer insertion in a physical feedthrough path when there is a disjoint voltage area even though, logically, the voltage areas belong to the same hierarchy. To enable this feature, set the `dct_enable_va_aware_ao_synthesis` variable to `true`.

For more information, see the *IC Compiler Implementation User Guide*.

Always-On Optimization on Top-Level Feedthrough Nets

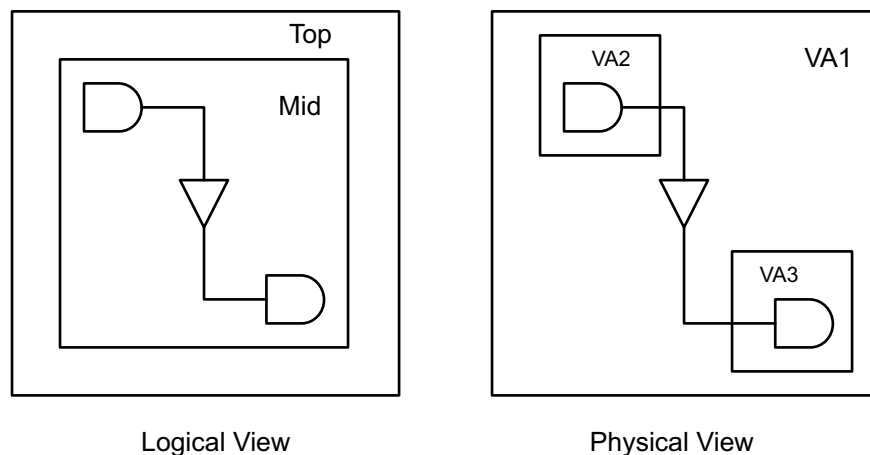
To perform always-on optimization on top-level feedthrough nets, you must specify the related supply net information on the output port that is driven by the feedthrough net. The Power Compiler tool derives the power and ground net information for the always-on buffers based on the domain's primary supply and related supply net that you specify for the output port driven by the feedthrough net. If the tool detects a level-shifter violation or an isolation violation on a feedthrough net, it sets a `dont_touch` attribute on the feedthrough net. This is done to prevent the shifting of the violation from one power domain to another.

Always-On Optimization on Disjoint Voltage Area

Power Compiler can insert always-on buffers on long nets that span physically distant voltage areas. Consider a long net as shown in [Figure 11-12](#). Logically, the net and the buffer are in the same hierarchy Mid, which is an always-on domain. However, physically, the net and the buffer are in two disjoint voltage areas.

If the library supports dual rail always-on cells, and the primary supply defined in the power domain for subdesign Mid is available in the power domain for Top, Power Compiler inserts dual rail always-on cells in the subdesign Mid that physically belongs to the Top design.

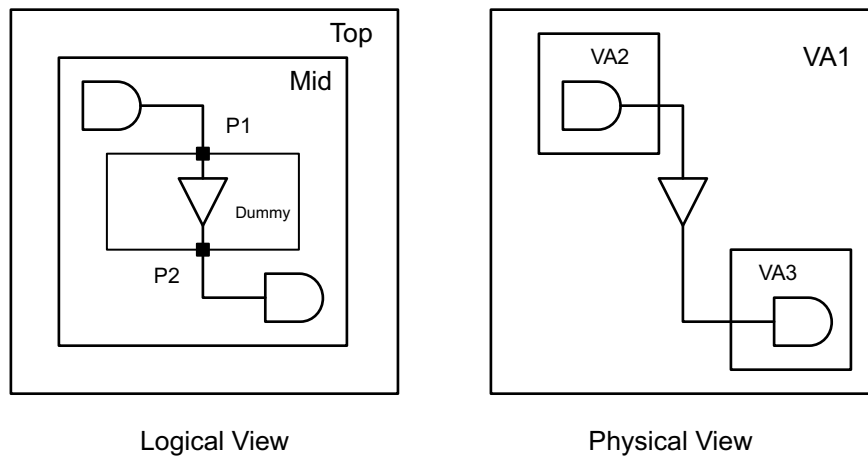
Figure 11-12 Always-On Buffer Insertion in Disjoint Voltage Areas



Power Compiler follows these steps to support always-on synthesis across disjoint voltage areas:

1. Create a dummy logic hierarchy inside the existing hierarchy Mid as shown in [Figure 11-13](#).
2. Create two hierarchical ports P1 and P2 on the dummy hierarchy and connect the buffer inside the dummy hierarchy to these ports.
3. Associate the dummy hierarchy to the already existing voltage area, to which the buffer belongs.

Figure 11-13 *Creating Dummy Hierarchy to Support Always-On Buffer Insertion in Disjoint Voltage Areas*



The creation of the dummy logic hierarchy and port punching on the dummy hierarchy allows the tool to perform always-on synthesis and legalization of always-on synthesis. The tool also supports associating the dummy hierarchy to the default voltage area, if the buffer belongs to the default voltage area.

Always-On Tie Cells

The Power Compiler tool appropriately chooses a normal tie cell, an always-on tie cell, or a combination of a single-power tie cell and a power management cell (isolation, level-shifter, or enable level shifter) to implement each tie-high and tie-low logic value specified in the design. This results in a more efficient implementation of tie-high and tie-low logic values at power domain boundaries.

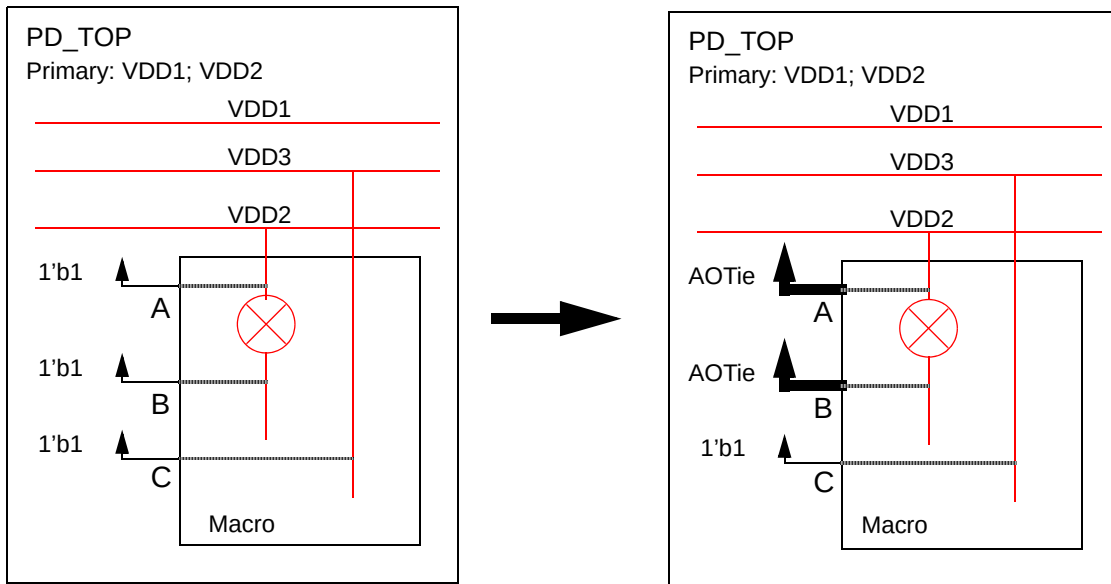
Basic Always-On Tie Cell Mapping

In general, all constants are assumed to be powered by the primary power supply of the domain where they reside. Therefore, each constant is mapped to a normal tie cell powered by its domain's primary power. However, if a constant is not driven by its primary supply during RTL simulation, it is mapped to an always-on (AO) tie cell. This enables the tool to insert tie cells on a constant path with logic using non-primary supplies.

However, in the case of constants driving macro cells that are powered by a supply different from its primary supply, these cells are assumed to be powered by the sink's related supply. If the sink's related supply is not available, for example, a macro cell's internal supply, an available supply that can drive the sink is selected. An AO or normal tie cell is used to map the constant powered by the selected supply.

In [Figure 11-14](#), the primary supply is VDD1, and VDD2 is an available supply. Pins A and B are related to an internal supply that is not available, so the tool finds another supply to drive those pins. Pins A and B are mapped to AO tie cells powered by a non-primary supply, VDD2, and pin C is left as a literal constant because VDD3 is not available in the power domain PD_TOP and VDD1 cannot drive it.

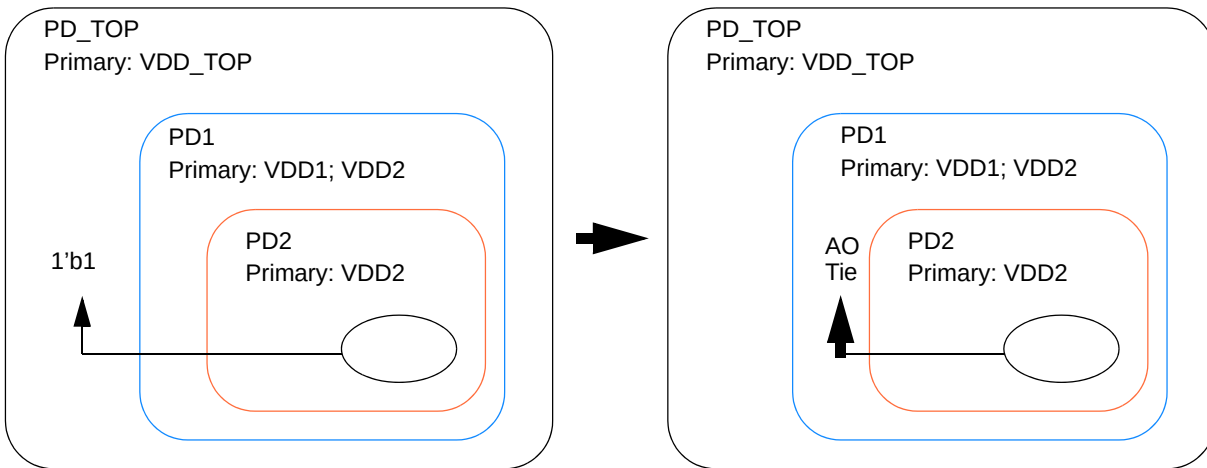
Figure 11-14 Macro Cells Using Non-Primary Supply



Enhanced Constant Propagation

Using an always-on (AO) tie cell for cell mapping increases the amount of instances that the constant can be propagated since these cells can be powered by a non-primary power supply. The example shown in [Figure 11-15](#) illustrates this point. The constant can be moved from PD_TOP to PD1 because VDD2 is an available supply. The constant cannot be moved to PD2 because boundary optimization and constant propagation are disabled on this domain. This example assumes that supply VDD2 is more on than VDD1 which is more on than VDD_TOP.

Figure 11-15 Enhanced Constant Propagation



Enhanced Always-On Tie Cell Mapping

The enhanced always-on tie cell mapping maps a literal constant according to the following rules:

1. If there is no load, a normal tie cell is mapped
2. If the primary supply can drive the load, a normal tie cell is mapped
3. If a non-primary supply is available that can drive the load, an always-on tie cell is mapped
4. Otherwise, the constant is left as a literal constant

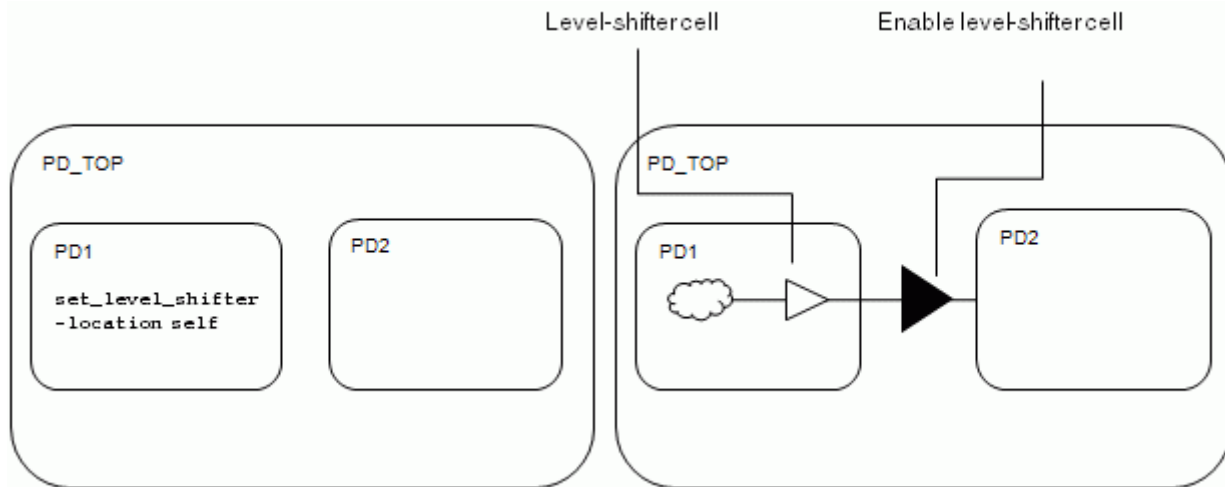
Specifying Level-Shifter Strategies

Use the `set_level_shifter` command to specify the strategy for inserting level-shifter cells between power domains that operate at different voltages. Level shifters are inserted by the tool during execution of the `compile_ultra` command. The level-shifter cells are inserted only on the power domain boundaries that operate at different voltages.

If a voltage violation exists at the domain boundary, the tool inserts level shifters at the domain boundary by default, even when a level-shifter strategy is not defined. This flexibility allows the tool to use the strategy that gets the best possible results. You can optionally restrict the insertion of level shifters to domain boundaries where the `set_level_shifter` command is used by setting the `upf_levshi_on_constraint_only` variable to `true`.

Specifying a strategy does not force a level-shifter cell to be inserted unconditionally. Power Compiler uses the power state table and the specified rules, such as threshold, to determine where level shifters are needed. When the tool identifies a potential voltage violation, it tries to resolve the violation by inserting multiple level-shifters or a combination of level-shifter and isolation cells. As shown in [Figure 11-16](#), when the tool finds a global net that has an isolation constraint, it inserts a level-shifter and an enable level-shifter cell, based on the voltage difference implied by the isolation power and isolation ground. The tool issues a warning message if it determines that a level shifter is not required.

Figure 11-16 Level-Shifter Insertion on Power Domain Boundaries



Use the `-elements` option to specify a list of ports and pins in the domain to which the strategy applies, overriding any `-threshold` or `-rule` settings. The `-no_shift` option prevents the insertion of level-shifter cells on the ports, pins, and nets specified by the `-elements` option. See [Specifying Design Instances Using Wildcard Characters](#) for information about the `-elements` option.

Use the `-name_prefix` or `-name_suffix` option of the `set_level_shifter` command to specify the naming of the level-shifter cell instances added during the implementation of a specific level-shifter strategy.

The following strategies have decreasing order of precedence, irrespective of the order in which they are executed:

```
set_level_shifter -domain domain_name -elements
set_level_shifter -domain domain_name -applies_to input/output
set_level_shifter -domain domain_name (with optional -applies_to both)
```

Use the `-update` option to add information to a level shifter strategy. You must use either the `-elements` or `-exclude_elements` option with the `-update` option.

Using Specific Library Cells With the Level-Shifter Strategy

When you specify a level-shifter strategy, by default the tool maps the level-shifter cells to a suitable level-shifter cell in the library. Use the `map_level_shifter_cell` command to limit the set of library cells to be used for the specified level-shifter strategy. This command does not force the insertion of the level-shifter cells. Instead, when the tool inserts the level-shifter cell, it chooses the library cells that are specified with the `-lib_cells` argument of the `map_level_shifter_cell` command. This command has no effect on instantiated level-shifter cells that have a `dont_touch` attribute set on them.

For more details, see the `map_level_shifter_cell` command man page.

Allowing Insertion of Level-Shifters on Clock Nets and Ideal Nets

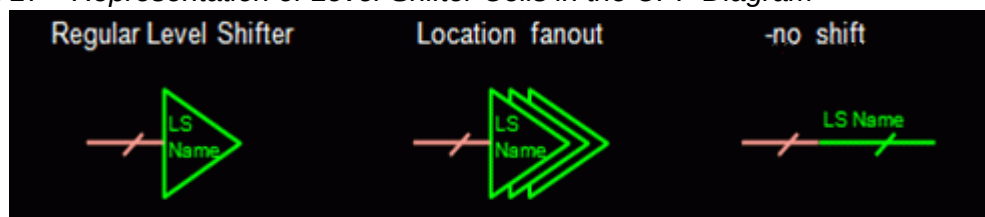
The Power Compiler tool does not insert level-shifter cells on clock nets, by default. Set the `auto_insert_level_shifters_on_clocks` variable to specific clock nets, for the tool to insert the level-shifter cells. Set this variable to `all`, for the tool to insert level-shifter cells on all clock nets that need level shifters.

Similarly, the Power Compiler tool does not insert level-shifter cells on ideal nets, by default. Set the `mv_insert_level_shifters_on_ideal_nets` variable to `all` for the tool to insert level-shifters on ideal nets. The default is an empty string ("").

Representing Level-Shifter Strategies in the UPF Diagram View

In the UPF diagram view, the level-shifter symbol is similar to a buffer symbol and includes a line segment representing the inputs that are shifted, as shown in [Figure 11-17](#). The location-fanout symbol looks similar to several buffers bundled together and indicates that the level-shifter cells occur on all fanout locations (sink) of the port that they are shifting. The no-shift symbol is represented by a line that shows the continuation of the inputs.

Figure 11-17 Representation of Level-Shifter Cells in the UPF Diagram

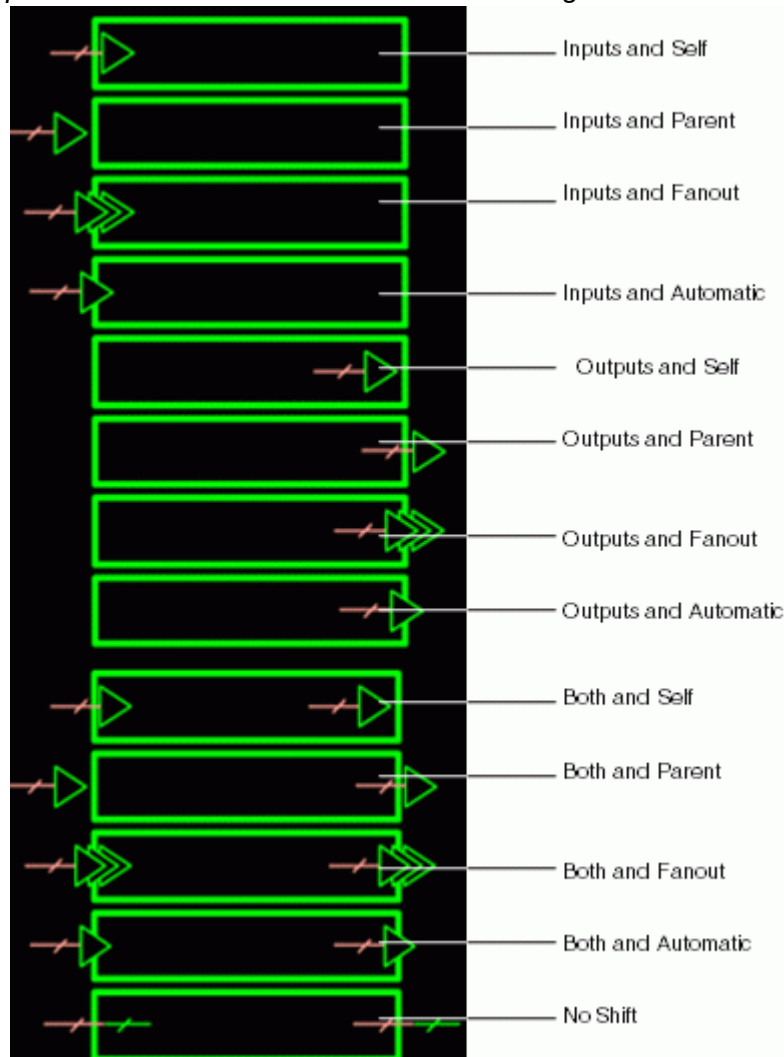


The symbol for each level-shifter strategy is located adjacent to the boundary of its parent power domain. The location depends on whether it shifts inputs or outputs.

Figure 11-18 shows the possible combinations of level-shifter symbols and locations, based on the values specified with the `-applies_to` and `-location` options of the `set_level_shifter` command.

The symbol appears at the left edge of the boundary if the strategy applies to input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports. If the strategy applies to both inputs and outputs, symbols appear at both left and right edges of the boundary.

Figure 11-18 Representation of Various Level-Shifter Strategies in the UPF Diagram



While defining the level-shifter strategy, if you specify the location as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

Note:

When you specify a list of elements to the level-shifter strategy using the `set_level_shifter -elements -applies_to` command, the UPF diagram positions the symbol relative to the left or right edge of the power domain boundary, based on whether the list contains input elements, output elements, or both.

Specifying Isolation Strategies

Use the `set_isolation` command to define the isolation strategy for a power domain and the elements in the power domain where the strategy is applied. The definition of an isolation strategy contains specification of the enable signal net, the clamp value, and the design elements where the strategy is applied.

The isolation power and ground nets must operate at the same voltage as the primary power and ground nets of the power domain where the isolation cells is located.

When you specify only the `-isolation_power_net` argument, the primary ground net is used as the isolation ground supply. Similarly, when you specify only the `-isolation_ground_net` argument, the primary supply net is used as the isolation power supply.

The `-isolation_supply_set` option specifies the power and ground functions of the same supply set to be used as the isolation power and isolation ground nets respectively. The `-isolation_supply_set` option is mutually exclusive with the `-isolation_power_net` and the `-isolation_ground_net` options.

The `-source` option filters the ports connected to a net that is driven by the specified supply set. When you use this option, the supply sets that are associated with each other are considered as connected.

The `-sink` option filters the ports driving a net that fans out to the logic driven by the specified supply set. Supply sets that are associated with each other are considered as connected.

When you specify both the `-source` and `-sink` options, isolation is applied to only those ports that have the specified source and sink.

The `-diff_supply_only` option determines the isolation behavior between the driver and the receiver supply sets or supply nets.

When the `-diff_supply_only` option is set to `true`, and the same supply set connects the driver and the receiver of a port on the interface of the reference power domain, the isolation cell is not added in the path from the driver to the receiver.

Note:

With the `-diff_supply_only` option, you can use the `-source` or the `-sink` option. You cannot use the `-source`, `-sink`, and `-diff_supply_only` options simultaneously.

The `-clamp_value` specifies the constant value in the isolation output: 0, 1, latch. The latch setting causes the value of the non-isolated port to be latched when the isolation signal becomes active.

Note:

Power Compiler does not support the value `z` for the `-clamp_value` option. The only supported values are 0, 1, and latch.

The `-elements` option specifies the elements for isolation, in cases where there are multiple isolation strategies within a given power domain. The specified elements can be input or output ports on the domain boundary, and design instances. The design instances must be the root cells of the power domain. The tool applies the isolation strategy only on domain boundaries and ignores the leaf cell instances.

When you specify the wildcard characters (`*` or `?`) with the `-elements` option, the tool searches for matching ports, pins, or design instances in the current level of hierarchy and applies the isolation strategy to the elements identified as the boundaries of the specified power domain. For details on the `-elements` option, see [Specifying Design Instances Using Wildcard Characters](#). To restrict the application of the isolation strategy on design instances, set the `upf_isols_allow_instances_in_elements` variable to `false`.

The tool filters the design elements, such as ports, pins, and design instances, when you specify the `-applies_to` option with the `-elements` option. To control the filtering behavior, set the `upf_iso_filter_elements_with_applies_to` variable. The valid values are `ERROR`, `ENABLE`, and `DISABLE`.

- **ERROR**
Generates an error message when you specify the `-applies_to` option with the `-elements` option.
- **ENABLE (the default)**
Filters the elements (pins, ports, and design instances) based on the value that you specify with the `-applies_to` option.
- **DISABLE**
Ignores the `-applies_to` option and applies the isolation strategy to all the elements specified with the `-elements` option. For the design instance specified with the `-elements` option, the isolation strategy is applied to all the pins of the specified instance.

When you do not specify any of the `-elements`, `-source`, and `-sink` options, the isolation strategy is applied to all the output ports of the power domain.

The `-no_isolation` option specifies that the elements in the `-elements` list should not be isolated. At least one of the `-isolation_power_net` or `-isolation_ground_net` or `-isolation_supply_set` arguments must be specified unless `-no_isolation` option is used.

Although the power state table can potentially reduce the number of isolation cells required, isolation synthesis is entirely based on directives set using the `set_isolation` and `set_isolation_control` commands.

The tool performs certain optimizations on isolation circuits, that do not affect the functionality. For example, if you have signals going from block A to block B, you specify output isolation on block A (in the parent) and input isolation on block B (in the parent). If the strategy results in two back-to-back isolation cells with no fan out in between, Power Compiler merges the isolation cells. It can merge the isolation cells based on the enable signal, power, or ground signals.

Use the `-name_prefix` or `-name_suffix` options of the `set_isolation` command to specify the naming of the isolation cell instances added during the implementation of a specific isolation strategy.

Use the `-update` option to add information to an existing isolation strategy. You must use this option with either the `-elements` or `-exclude_elements` option.

Every isolation strategy defined by a `set_isolation` command must have a corresponding `set_isolation_control` command, unless the strategy is defined with the `-no_isolation` option.

Using the `set_isolation_control` Command

The `set_isolation_control` command specifies the isolation control signal and isolation sense separately. The command identifies an existing isolation strategy and specifies the isolation control signal for that strategy.

Using the location value you specify with the `-location` option of the `set_isolation_control` command, Power Compiler identifies isolation cells in the power domain across the design hierarchy and associate them with UPF strategies. When the value you specify is `self`, the tool starts the search from the port on the boundary of the power domain, and traverses inside the power domain until it encounters either a cell, a multiple fanout net, or the boundary of another power domain. When the location you specify is `parent`, the tool starts the search from the port on the boundary of the power domain, and traverses outside the power domain until it encounters a cell, a multiple fanout net, or the boundary of another power domain. When the value you specify is `fanout`, the isolation cells are inserted at all fanout locations of the port being isolated, and inside the power domain of respective loads. For more information about the rules that apply when you use the `-location fanout` option, see [Rules Applicable for Location Fanout](#).

When the tool encounters an isolation cell that is not already associated with an isolation strategy, it associates the cell with an appropriate isolation strategy. This association is based on the values you specified with the `-clamp_value` option of the `set_isolation` command, and the `-isolation_sense` option of the `set_isolation_control` command.

If the cell encountered is not an isolation cell, the tool does not treat the port as an isolation port, and during the next optimization step, the tool inserts an isolation cell.

The `-isolation_sense` option specifies the logic state of the isolation control signal that places isolation cells in the isolation mode. The possible values for this option are 0 or 1. The default is 1. The isolation signal specified by the `-isolation_signal` option can be for a port or pin or a net, with the port/pin having higher precedence. The isolation signal need not exist in the logic hierarchy where the isolation cells are to be inserted; the synthesis or implementation tool can perform port-punching as needed to make the connection. Port-punching means automatically creating a port to make a connection from one hierarchical level to the next. These punched ports are not considered for isolation or level-shifting, even though after the port creation, these ports reside within the coverage of an isolation or level-shifter strategy.

Existing ports are isolated and level-shifted according to the applicable isolation and level-shifter strategy, even if they reside on an always-on path.

Rules Applicable for Location Fanout

Follow these rules while using the `set_isolation` and `set_isolation_control` command, when you use the `-location fanout` option:

- Do not use the `-isolation_power_net` option of the `set_isolation` command, when you use the `-location fanout`. However, when you use the `-location` option with the value `parent` or `self`, you can use the `-isolation_power_net` option of the `set_isolation` command.
- The `-location fanout` option can be used only when you use one of the `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command. Similarly, when you use the `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option, you can only specify fanout with the `-location` option.
- The `-no_isolation` option cannot be used when you use `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command.
- Set the `derived_iso_strategy` attribute, using the `set_design_attributes` command, before specifying `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command. After setting the `derived_iso_strategy`, if you do not specify `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command, the tool issues an error.
- If you set the `derived_iso_strategy` attribute, the only value that you can specify with the `-location` option is `fanout`.

For more details about setting the UPF attributes on ports and hierarchical cells, see [Setting UPF Attributes on Ports and Hierarchical Cells](#)

Order of Precedence of Isolation Strategies

The isolation strategies have the following decreasing order of precedence, irrespective of the order of execution. The tool uses this order of precedence while querying the isolation strategy on a pin.

1. Pin or port-level strategy matching the `-no_isolation` option.
2. Pin or port-level strategy matching the `-source` and `-sink` options.
3. Pin or port-level strategy matching the `-source` and `-diff_supply_only` options on an input pin or matching the `-sink` and `-diff_supply_only` options on an output pin.
4. Pin or port-level strategy matching the `-sink` and `-diff_supply_only` options on an input pin or matching the `-source` and `-diff_supply_only` options on an output pin.
5. Pin or port-level strategy matching the `-source` option on an input pin or matching the `-sink` option on an output pin.
6. Pin or port level strategy matching the `-sink` option on an input pin or matching the `-source` option on an output pin.
7. Pin or port-level strategy matching the `-diff_supply_only` option and without the `-source` and `-sink` options.
8. Pin or ports specified with the `-elements` option, without specifying the `-source`, `-sink`, or `-diff_supply_only` option.
9. Cell-level strategy with the `-no_isolation` option.
10. Cell-level strategy matching the `-source` and `-sink` options.
11. Cell-level strategy matching the `-source` and `-diff_supply_only` options on an input pin, or matching the `-sink` and `-diff_supply_only` options on an output pin.
12. Cell-level strategy matching the `-sink` and `-diff_supply_only` options on an input pin, or matching the `-source` and `-diff_supply_only` options on an output pin.
13. Cell-level strategy matching the `-source` option on an input pin, or matching the `-sink` option on an output pin.
14. Cell-level strategy matching the `-sink` option on an input pin, or matching the `-source` option on an output pin.
15. Cell-level strategy specified with the `-diff_supply_only` option and without specifying the `-source` or `-sink` options.
16. Cell-level strategy specified without using the `-source`, `-sink`, or `-diff_supply_only` options.
17. Domain-level strategy with the `-no_isolation` option.

18. Domain-level strategy matching the `-source` and `-sink` options.
19. Domain-level strategy matching the `-source` and `-diff_supply_only` options on an input pin, or matching the `-sink` and `-diff_supply_only` options on an output pin.
20. Domain-level strategy matching the `-sink` and `-diff_supply_only` options on an input pin, or matching the `-sink` and `-diff_supply_only` options on an output pin.
21. Domain-level strategy matching the `-source` option on an input pin or matching the `-sink` option on an output pin.
22. Domain-level strategy matching the `-sink` option on an input pin or matching the `-source` option on an output pin.
23. Domain-level strategy specified with `-diff_supply_only` option and without the `-source` or `-sink` options.
24. Domain-level strategy specified without the `-source`, `-sink`, and `-diff_supply_only` options.

Using Specific Library Cells With Isolation Strategies

When you define an isolation strategy, by default the tool associates the isolation strategy with any suitable isolation cell in the library. When the library does not contain a complete set of isolation cells, you can use some of the basic gates as isolation cells. For more information, see [Multivoltage Design Concepts](#).

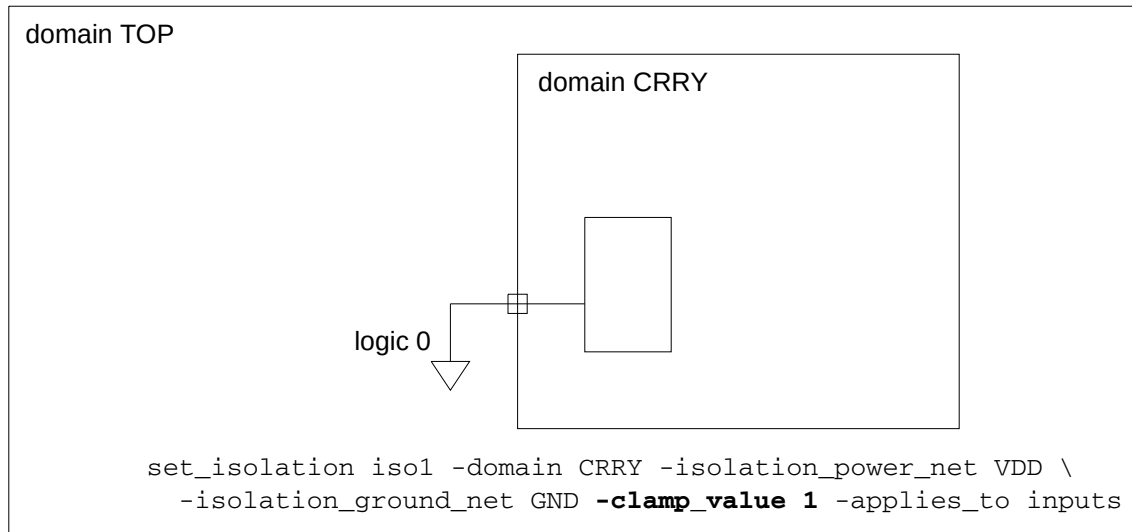
To associate a specific set of library cells with the isolation strategy, use the `map_isolation_cell` command. The `map_isolation_cell` command can also be used to associate standard cells used as isolation cells with the isolation strategy.

When designs contain instantiated isolation cells that are associated with an isolation strategy, the `map_isolation_cell` command remaps these library cells to the cells specified with the `-lib_cells` argument of the command. If the instantiated isolation cells have `dont_touch` attribute set on them, the command does not remap these cells. The command has no impact on the instantiated isolation cells that are not, or cannot be associated with an isolation strategy.

For more information, see the command man page.

Aligning Isolation Strategies to Constant Drivers

Consider a situation where a driver forces a port to a constant value, either a logic 0 or 1, at a power domain boundary, but the isolation clamp value defined for that port in the UPF file is the opposite value, as shown in the example in [Figure 11-19](#).

Figure 11-19 Constant Driver and Conflicting Isolation Strategy

This situation can arise during logic optimization when Power Compiler moves or splits a constant value that crosses domain boundaries. The tool might need to insert an isolation cell to prevent a formal verification error. In such a situation, you might want to modify the isolation strategy to match the constant value so that the isolation cell can be removed during optimization.

To automatically generate new UPF isolation commands that are consistent with the constant driver values, you can use the `generate_mv_constraints` command, as shown in the following example. Suppose that the original UPF commands define an isolation strategy named `iso1`, which applies to the inputs of the `CRRY` power domain and sets a clamp value of 1 as follows:

```
dc_shell> set_isolation iso1 -domain CRRY -isolation_power_net VDD \
-isolation_ground_net GND -clamp_value 1 -applies_to inputs
dc_shell> set_isolation_control iso1 -domain CRRY \
-isolation_signal ctrl -isolation_sense low -location self
```

However, the design has a constant driver element driving the `u1/cin` pin with a value of 0 at the boundary of `CRRY` power domain, which is a conflict with the isolation strategy. To generate a new isolation strategy to match the constant driver, use this command:

```
dc_shell> generate_mv_constraints -align_isolation_clamp_value \
-output align.upf
```

The command detects the conflict between the isolation strategy and the constant driver and generates the following new isolation strategy commands to resolve the conflict:

```
# Created by DC Utility for non-matching isolation clamp and driving
# constant value.
# List of new strategies created with clamp value matching the constant
# isolating it, for strategies with no user specified element list
set_isolation iso1_clamp0 -domain CRRY \
  -isolation_power_net VDD -isolation_ground_net GND \
  -elements u1/cin -clamp_value 0 -applies_to inputs
set_isolation_control iso1_clamp0 -domain CRRY \
  -isolation_signal ctrl -isolation_sense low -location self
```

The `generate_mv_constraints` command writes comments and the two new isolation strategy commands to the `align.upf` file. It creates the name of the new isolation strategy by appending the `_clamp0` suffix to the original strategy name. The new `set_isolation` command always uses the `-elements` option to identify the pins to which the strategy applies. The new strategy is more specific than the original strategy, so it has higher priority.

When using the `generate_mv_constraints` command, specify the `-output` option to write the commands into a file, the `-apply` option to execute the new commands, or both options to perform both actions. If you use the `-apply` option, the newly created and modified strategies are applied to the design in memory, and any subsequent usage of the `save_upf` command writes the new isolation commands along with the original UPF commands.

By default, the `generate_mv_constraints -align_isolation_clamp_value` command checks only for conflicts involving general `set_isolation` strategies specified without the `-elements` option, for example, using `-applies_to inputs`. To also fix conflicts with isolation strategies specified with the `-elements` option of the `set_isolation` command, use the `-include_elements` option of the `generate_mv_constraints` command.

For example, suppose that the original UPF commands define an isolation strategy named `iso2`, which applies to the `u1/cin` input and `u1/carry` input pins of `CRRZ` power domain and sets a clamp value of 1 for these pins, as follows:

```
dc_shell> set_isolation iso2 -domain CRRZ -isolation_power_net VDD \
  -isolation_ground_net GND -elements {u1/cin u1/carry} \
  -clamp_value 1 -applies_to inputs
dc_shell> set_isolation_control iso2 -domain CRRZ \
  -isolation_signal ctrl -isolation_sense low -location self
```

However, the design has a constant driver element driving the `u1/cin` pin with a value of 0 at the boundary of the `CRRZ` power domain, which is a conflict with the isolation strategy. To fix the element-level strategy for conflicts with constant drivers, use the `generate_mv_constraints` command with the `-include_elements` option as follows:

```
dc_shell> generate_mv_constraints -align_isolation_clamp_value \
  -include_elements -output align.upf -apply
```

The command detects the element-level conflict and generates the following new strategies to resolve the conflict:

```
# Created by DC Utility for non-matching isolation clamp and driving
# constant value.
# List of user strategies modified
set_isolation iso2_modified -domain CRRZ -isolation_power_net VDD \
  -isolation_ground_net GND -elements u1/carry -clamp_value 1 \
  -applies_to inputs
set_isolation_control iso2_modified -domain CRRZ -isolation_signal ctrl \
  -isolation_sense low -location self

# List of new strategies created with clamp value matching the constant
# isolating it, for strategies with user specified element list
set_isolation iso2_clamp0 -domain CRRZ -isolation_power_net VDD \
  -isolation_ground_net GND -elements u1/cin -clamp_value 0 \
  -applies_to inputs
set_isolation_control iso2_clamp0 -domain CRRZ -isolation_signal ctrl \
  -isolation_sense low -location self
```

The `generate_mv_constraints` command generates two new strategies: one named `iso2_modified` for the input without a conflict and another one named `iso2_clamp0` for the input that has a conflict. When you use the `-apply` option, the new strategies are applied to the design in memory and the original isolation strategies are updated to remove the elements that are listed in the new isolation strategy; any subsequent usage of the `save_upf` command writes out the new isolation commands.

If you use the `generate_mv_constraints` command, you must do so before you compile the design. If the netlist already contains isolation cells, using the `generate_mv_constraints` command might result in back-to-back isolation cells or loss of association between existing isolation cells and the isolation strategy.

For more information, see the `generate_mv_constraints` command man page.

Isolation and Level-Shifter Cells Connected Back-to-Back

Power Compiler supports eight different combinations of isolation and level-shifter cells connected back to back, on the same side of the power domain boundary. It is required that the source power is more or equally always-on than the destination power. When an enable

level-shifter cell is available in the library, the tool replaces the level-shifter and isolation cell combination with an enable level-shifter cell. [Table 11-3](#) shows each of the combinations of the isolation and level-shifter locations supported by Power Compiler.

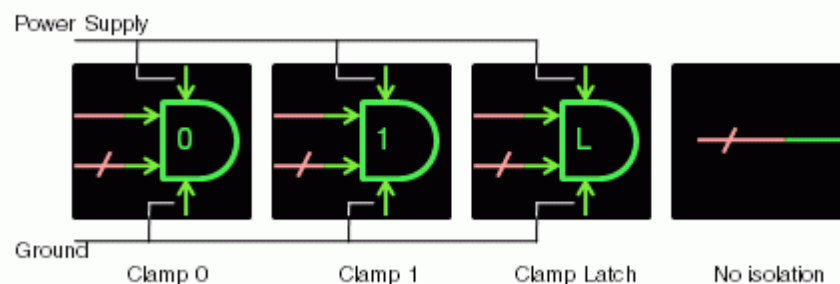
Table 11-3 *Combination of Isolation and Level-Shifter Cells Connected Back-to-Back*

Target power domain	Isolation location	Level-shifter location	Isolation power	Replaced by an enable level-shifter cell?
source	self	self	source	no
source	self	self	destination	yes
source	parent	parent	source	no
source	parent	parent	destination	yes
destination	parent	parent	source	no
destination	parent	parent	destination	yes
destination	self	self	source	no
destination	self	self	destination	yes

Representing Isolation Strategies in the UPF Diagram View

[Figure 11-20](#) shows the symbols used to represent an isolation strategy in the UPF diagram view. The symbol used is similar to an AND gate and the clamp value is shown inside the symbol. The symbol also includes pins for power and ground, a segment representing the isolation signal, and a line segment representing the inputs or outputs that the strategy isolates. When the `-no_isolation` option is specified, a straight line is used to show the continuation of the inputs.

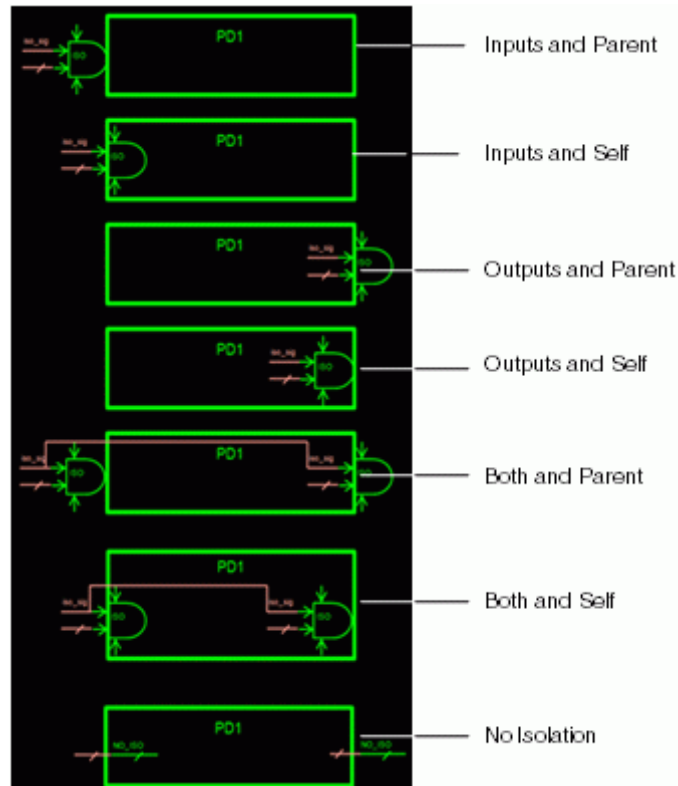
Figure 11-20 *Representation of Various Types of Isolation Cells in the UPF Diagram*



The symbol is located adjacent to the boundary of its parent power domain. The location also depends on whether the strategy isolates inputs or outputs.

Figure 11-21 shows all possible combinations of isolation strategy symbols and locations, based on the value of the `-applies_to` option of the `set_isolation` command and the value of the `-location` option of the `set_isolation_control` command used in defining isolation strategy.

Figure 11-21 Representation of Various Types of Isolation Strategies in the UPF Diagram



The symbol appears to the left edge of the power domain boundary if the strategy applies to the input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports.

If the strategy applies to both input and output ports, the symbol appears at both left and right edges of the boundary.

While defining the isolation strategy, if you specify the location as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

Note:

If you specify a list of elements using the `set_isolation -elements` command, the UPF diagram ignores the `-applies_to` option and positions the isolation symbol relative

to the left or right edge of the power domain boundary, based on whether the list contains input elements or output elements or both.

Setting UPF Attributes on Ports and Hierarchical Cells

You can set attributes on specific ports and hierarchical cells of a power domain to specify additional requirements.

Setting Attributes on Ports

To set attributes and their values on the specified ports, use the `set_port_attributes` command. [Table 11-4](#) shows the list of attributes and their values that can be specified on the ports using the `set_port_attributes` command.

Table 11-4 The UPF Port Attributes

Attribute name	Attribute value	Ports where the attribute can be specified	Use of the attribute
<code>model</code>	Name of the object		Specifies a module or library cell to whose ports the attribute is to be applied.
<code>iso_sink</code>	Name of the supply set, <code>DERIVED_DIFF_ONLY</code> , <code>DERIVED_DIVERSE</code>	Output	Identify the actual off chip load of a primary output port.
<code>iso_source</code>	Name of the supply set <code>DERIVED_DIFF_ONLY</code> , <code>DERIVED_DIVERSE</code>	Input	Identify the actual off chip driver of a primary input port.
<code>related_supply_default_primary</code>	Boolean	Top level input and output ports	Indicates that, when the related supply net is not specified, the primary supply of TOP domain is assumed as the related supply. Used by the verification tools so that no assumption is made about the default power supply.

Table 11-4 The UPF Port Attributes (Continued)

Attribute name	Attribute value	Ports where the attribute can be specified	Use of the attribute
snps_derived	Boolean	Input and output supply ports	Indicates that the specified ports have been created by Synopsys tools. You can specify this attribute in the bottom-up flow as well.
repeater_power_net repeater_ground_net	Name of the supply net	Input and output ports and pins. Cannot be specified on bidirectional ports	Tool inserts a repeater (buffer) to drive the specified port. The inserted buffer is powered by the specified supply net.
feedthrough		Input and output ports and pins.	Indicates that the specified ports are connected together internally to form a feedthrough. You must use this option with the -model and -ports options. This option is mutually exclusive to the -unconnected option.
unconnected		Input and output ports and pins.	Indicates that the specified ports are not internally connected. You must use this option with the -model and -ports options. It is mutually exclusive to the -feedthrough option.
clamp_value	1, 0, or latch	Input and output ports and domain boundary pins.	Specifies the clamp value if the port has an isolation strategy associated with it. This option sets the UPF_clamp_value attribute.

Note:

When you specify the `set_port_attributes` command multiple times on the same object, the last setting overrides the previous settings.

The following example shows how to set the `iso_source` and `iso_sink` attributes on the input and output ports, respectively.

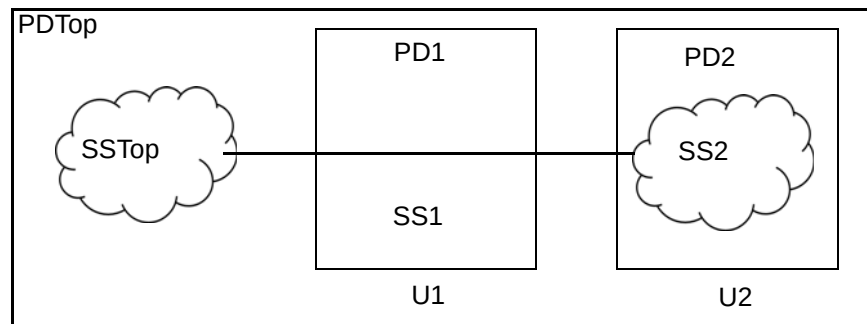
```
dc_shell> set_port_attributes -ports {in1 in2} -attribute iso_source SS1
dc_shell> set_port_attributes -ports {out1 out2} -attribute iso_sink SS2
```

Specifying Supplies for Repeaters

Repeaters are buffers inserted at regular intervals along the length of a long net to maintain sufficient drive strength along the full length of the net. In Design Compiler, the `insert_buffer` command lets you specify the number of such buffers to insert into a net by using the `-no_of_cells` option.

If a long net crosses a power domain boundary, such as in the case of a feedthrough net crossing through a power domain, repeater buffers inserted inside the power domain must, by default, maintain the always-on characteristics of the sink domain. For example, in [Figure 11-22](#), PDTop power domain is more always-on than PD2 power domain, and PD2 power domain is more always-on than PD1 power domain. The feedthrough path through PD1 power domain must be always-on with respect to PD2 power domain.

Figure 11-22 Feedthrough Path in PD1 Power Domain Before Buffer Insertion



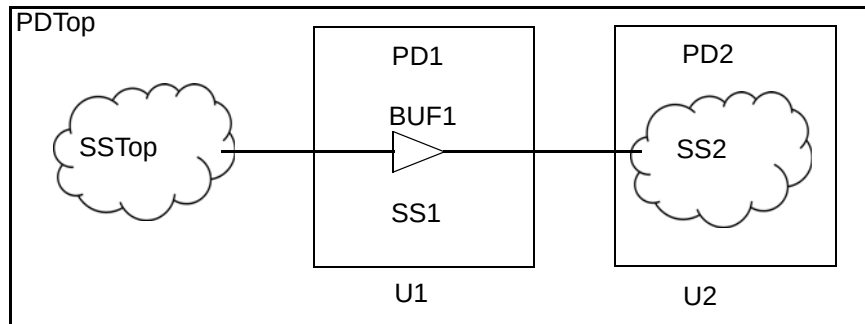
However, the always-on requirement might not be needed in certain cases. For example, if the feedthrough net is a DFT scan signal that is used only when all power domains are active, the inserted buffers can use the PD1 power supply, thereby using less resources. In other cases, depending on the floorplan, the power supplies of the sink domain might not be easily available where the buffer needs to be inserted.

To enforce the insertion of a buffer with a specific power supply on the feedthrough path, use the `-repeater_supply` argument of the `set_port_attributes` command. For example,

```
dc_shell> set_port_attributes -elements {U1} \
    -applies_to outputs -repeater_supply SS1
```

The tool inserts repeater buffers that drive the output port of elements U1 and uses the supply set SS1 to power these buffers. This results in the buffering shown in [Figure 11-23](#).

Figure 11-23 Feedthrough Path in PD1 Power Domain With a Buffer Inserted



Insertion of the repeater buffers might cause the need for additional level shifter and isolation cells on the feedthrough path.

When you specify the insertion of a repeater buffer to drive a specified port, you must also specify the power supply for the buffer. The supply specified with the `repeater_supply` attribute must be available in the scope of the power domain where the buffer is inserted.

You can specify either a supply set using the `-repeater_supply` option as shown in the previous example, or a pair of supply nets (power and ground) using the `-attribute` option, as shown in the following example:

```
dc_shell> set_port_attributes -elements {U1} -applies_to outputs \
          -attribute repeater_power_net VDD \
          -attribute repeater_ground_net VSS
```

You must specify the `-attribute` option two times in the same command, to specify the power and ground nets for the inserted repeater buffers. Multiple occurrences of the `-attribute` option are allowed only for the `repeater_power_net` and `repeater_ground_net` attributes.

The repeater insertion is performed by the `compile_ultra`, `insert_mv_cells` or the `insert_dft` command, before inserting other power management cells. The tool inserts either a single non-inverting repeater (buffer) or a pair of inverters. You cannot specify the type of the repeater to be inserted.

After inserting the repeater, during level-shifter and isolation cell insertion, the tool ensures that the repeater insertions do not cause electrical violations and inserts a level-shifter or an isolation cell to fix the violation. However, you must have defined an isolation strategy for the tool to insert the isolation cell. The `check_mv_design` checks and reports any violation introduced and not fixed by the repeater insertion.

If the repeater does not use the primary supply of the domain as the supply, the `save_upf` command writes the `connect_supply_net` command for the PG pins of the repeater.

For more information, see the `set_port_attributes` command man page.

Setting Attributes on Hierarchical Cells

To set attributes on a collection of cells, use the `set_design_attributes` command.

[Table 11-5](#) shows the list of attributes and their values that can be specified on hierarchical cells using the `set_design_attributes` command.

Table 11-5 The UPF Design Attributes

Attribute name	Attribute value	Location of the attribute	Use of the attribute
<code>derived_external</code> and <code>external_supply_map</code>	Name of the supply set	Hierarchical cell	Indicates that the supply sets are reference-only supply sets. These are used with ports with the <code>iso_source</code> and <code>iso_sink</code> attributes. These attributes establish a one-to-one order dependent mapping of the supply sets.
<code>derived_iso_strategy</code>	Name of the isolation strategy	Hierarchical cell	To ensure unique strategy name for the derived strategies in the power domain. Used in hierarchical flow to support location fanout
<code>enable_bias</code>	Boolean	Top-level design	When set to <code>true</code> , turns on the well bias feature.
<code>merge_domain</code>	Boolean	Hierarchical cell that is not the root cell of the power domain	Indicates that the two blocks belonging to the same power domain can be merged
<code>lower_domain_boundary</code>	Boolean	Top scope of the design and any hierarchical cell	When set to <code>true</code> , the boundary of the power domain extends up to the boundary of the power domain below it.
<code>suppress_iss</code>	Power domain	Current design	Indicates that supply set handles cannot be created in the power domain
<code>upf_chip_design</code>	Boolean	Top-level design	Indicates that the design is the TOP block. When the isolation strategy definition contains <code>-location fanout</code> , this attribute causes the primary output ports to be considered as the loads

Table 11-5 The UPF Design Attributes (Continued)

Attribute name	Attribute value	Location of the attribute	Use of the attribute
correlated_supply_group	Supply net names or wildcard (*) character	Top scope of the design	Indicates that the supply nets of the port state or power state triplets should be considered as correlated voltage range
legacy_block	Boolean	Hierarchical cell	When set to true, indicates that the block is a legacy block. This is useful while combining two blocks; one defined using domain-dependent supply nets and the other defined using domain-independent supply nets and supply sets. The block defined using domain-dependent supply nets is marked as a legacy block.

Extending the Power Domain Boundary

The Power Compiler tool considers the logic boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801-2009 standard, the tool can extend the power domain boundary to include the boundary of another power domain contained in it. The strategies defined for a power domain apply to the pins and other objects on the lower domain boundary.

To extend the power domain boundary to the boundary of another power domain contained in it, set the `lower_domain_boundary` attribute on the design, as shown in the following example:

```
set_design_attributes -elements {.} -attribute lower_domain_boundary true
```

You must enable or disable this feature for the entire design. If you selectively enable or disable the feature for a few blocks, the tool issues error messages.

For more details about the lower domain boundary feature, see [Appendix A, “Lower Domain Boundary Support.”](#)

Setting Terminal Boundaries

When you set the design attribute `terminal_boundary` to true on an instance, the boundary ports of this instance become a terminal boundary. For example, to set a terminal boundary on an instance named U1, do the following:

```
set_design_attributes -elements U1 -attribute terminal_boundary true
```


When you use the `terminal_boundary` attribute, only the root cells of a power domain are allowed in the element list. These cells can be macro cells, block abstracts, black boxes, and other hierarchies.

If you specify a cell that is not a root cell, the tool issues a UPF-210 error message.

When you set the `terminal_boundary` attribute to `true` on a particular block, the tool considers its boundary constraints for performing supply consistency checks. There is no other functional implication of this attribute. The `terminal_boundary` option is used in a bottom-up hierarchical implementation flow. In a flat design, the tool ignores any `terminal_boundary` attribute specified on a nested domain's root cells.

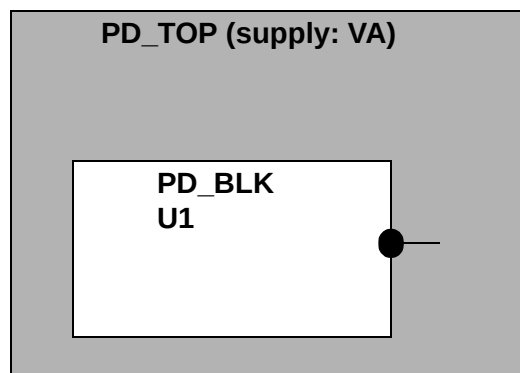
For example, assume you use the following command on the block named U1 shown in [Figure 11-24](#):

```
set_design_attributes -elements U1 -attribute terminal_boundary true
```

The tool behaves as follows:

- If the block U1 is a .ddc file or a block abstract, during top synthesis, the tool ignores the `terminal_boundary` specified on U1.
- If the block U1 is an ETM or macro, the tool retains the `terminal_boundary` attribute setting on block U1 in the top-level integrated UPF.
- If the block U1, is a black box, the tool retains the `terminal_boundary` attribute setting in the top-level integrated UPF, and it performs a consistency check between the boundary constraint at U1's port and the external driver or load cell supply.

Figure 11-24 Example of Hierarchical Terminal Boundary



Specifying Retention Strategies

The `set_retention` and `set_retention_control` commands specify the strategy for inserting retention cells inside the power-down domains.

The `set_retention` command specifies which registers in the power-down domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality.

When you do not specify the `-elements` option, the retention strategy is applied to all sequential cells in the power domain, unless you specify the `-no_retention` option. For details on the `-elements` option, see [Specifying Design Instances Using Wildcard Characters](#). Note that DesignWare instances are supported when using the `-elements` option with the `set_retention` command. Power Compiler applies the `size_only` attribute on all the elements on which it applies the retention strategy.

The `-retention_power_net` and `-retention_ground_net` options specify the supply nets to be used as the retention power and ground nets. The retention power and ground nets are automatically connected to the implicit save and restore processes and shadow register. If you specify only the `-retention_power_net` option, the primary ground net is used as the retention ground supply. If you specify only the `-retention_ground_net` option, the primary supply net is used as the retention power supply.

The `-retention_supply_set` option specifies the supply set whose power and ground functions have to be associated as the retention power and retention ground nets respectively. If you specify the `-retention_supply_set` option, the power and ground functions of the same supply set should be used as the retention power and retention ground nets respectively.

When specific objects in the power domain do not require retention capabilities, you can specify them with the `-no_retention` option. Power Compiler maps these objects to library cells that do not have retention capability or functionality.

The `-save_condition`, `-restore_condition`, and `-retention_condition` options are intended to capture the clock-dependent retention behavior during simulation. These options are parsed and ignored by Power Compiler, but they are preserved and written out by the `save_upf` command if the netlist is not synthesized.

The following strategies have decreasing order of precedence, irrespective of the order in which they are executed:

```
set_retention -domain -elements
set_retention -domain
```

The power and ground nets of the retention registers can operate at voltage levels different from the primary and ground supply voltage levels of the power domain where the retention cell is located.

Every retention strategy defined without the `-no_retention` option, must have a corresponding `set_retention_control` command. The `set_retention_control` command specifies the retention control signal and retention sense. The command identifies an existing retention strategy and specifies the save and restore signals and senses for that strategy.

Each control signal can be a port, pin or a net, with a port or pin having higher precedence. The retention signal does not need to exist in the logic hierarchy where the retention cells are to be inserted. The synthesis or implementation tools perform port-punching, as needed, to make the connection. Port-punching automatically creates a port to make a connection from one hierarchical level to the next. These punched ports are not considered for isolation, even though after the port creation, these ports reside within the coverage of an isolation strategy.

The `-update` option allows you to refine the element list of a previously defined retention strategy. When used with the `-elements` option, the set of elements is the union of all elements specified for a strategy. You cannot refine a domain-based retention strategy to an element-based retention strategy with the `-update` option. [Example 11-5](#) illustrates this case. The second `set_retention` command (highlighted) results in an error.

Example 11-5 Refining Domain-Based Retention Strategy to Element-Based Strategy

```
create_power_domain MID -elements {mid1 mid2}
set_retention RET1 -domain MID
set_retention_control RET1 ...
map_retention_cell RET1 ...
set_retention RET1 -domain MID -elements {mid1} -update
```

The `-assert_r_mutex`, `-assert_s_mutex`, and `-assert_rs_mutex` options are intended to capture the clock-dependent retention behavior during simulation. These options are parsed and ignored by Power Compiler.

The `set_retention_elements` command defines a list of critical elements that can later be used in a `set_retention` command. The list of elements applies to the scope where the `set_retention_elements` is defined. The list of elements must either all be retained or none of them are retained. It is an error to have a partially retained list of elements.

Choosing Specific Library Cells With Retention Strategies

The `map_retention_cell` command provides a mechanism for constraining the implementation choices for retention registers. The command must specify the name of an existing retention strategy and power domain.

The `-lib_cell_type` option directs the tool to select a retention cell that has the specified cell type. However, the value specified with this option does not change the simulation semantics specified by the `set_retention` command.

The `-lib_model_name` option specifies the retention register verification model in the input UPF file, which is parsed for syntax. The model information is primarily used by the Formality tool. The model information is not captured in the compiled database, `.ddc` file or in the UPF file written after synthesis.

The `retention_cell` attribute on the library cells in the target library defines the retention styles of the library cells.

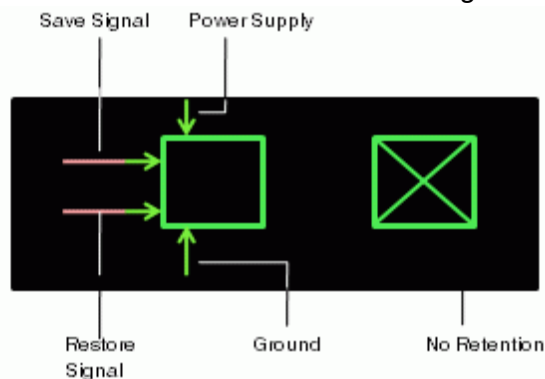
Retention Strategy and Clock-Gating Cells

When you define retention strategy for a power domain, by default, Power Compiler does not apply the retention strategy to the clock-gating cells in the power domain. The tool does not issue warning or information message. However, if you set the `upf_use_additional_db_attributes` variable to `false`, the tool issues a UPF-117 warning message for every power domain defined with a retention strategy and contains clock-gating cells. Formal verification also flags a failure in this situation.

Representing Retention Strategies in the UPF Diagram View

In the UPF diagram view, the retention cell is represented by a green bounding box as shown in [Figure 11-25](#). The symbol includes pins for power and ground and segments for save and restore signals. The no-retention symbol contains a “X” inside the bounding box.

Figure 11-25 Representation of Retention Cells in the UPF Diagram



All retention symbols are located at the center of their parent power domains. The diagram displays the supply nets connected to the retention strategy, the domains to which the strategy belongs and their save and restore signals.

Creating Power Switches

The `create_power_switch` command creates a virtual instance of power switch in the scope of the specified power domain. The power switch has at least one input supply port and one output supply port. When the switch is off, the output supply port is shut down and has no power.

The `create_power_switch` command lets the tool know that a generic power switch resides in the design at a specific scope or level of hierarchy. The off state of the power switch output is used in the power state table. Power Compiler does not perform power switch insertion, but the information is passed to IC Compiler for implementation.

The following power switch definition is for the power switch in [Figure 11-1](#).

```
create_power_switch SW1 \
  -domain PD_TOP \
  -output_supply_port {SWOUT VDD1g} \
  -input_supply_port {SWIN1 VDD1} \
  -control_port {CTRL swctl} \
  -on_state {ON VDD1 {!swctl}}
```

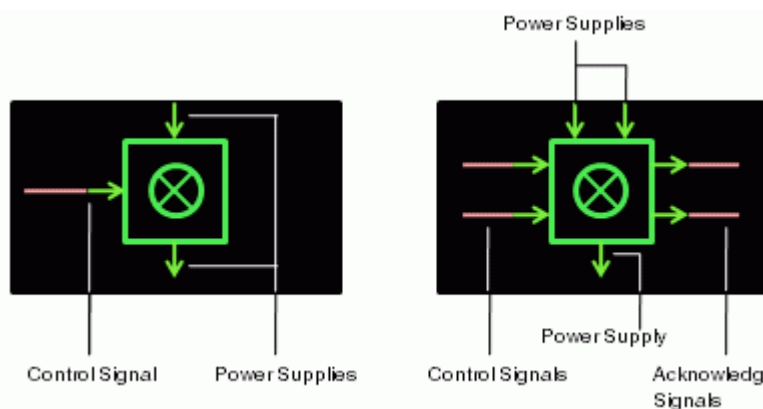
The UPF standard requires a simple name for the `switch_name` argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to true.

For more information, see the `create_power_switch` command man page and the *Synopsys Multivoltage Flow User Guide*.

Representation of Power Switches in the UPF Diagram View

In the UPF diagram view, a power switch is represented by a green colored circle with a “X” inside it, as shown in [Figure 11-26](#).

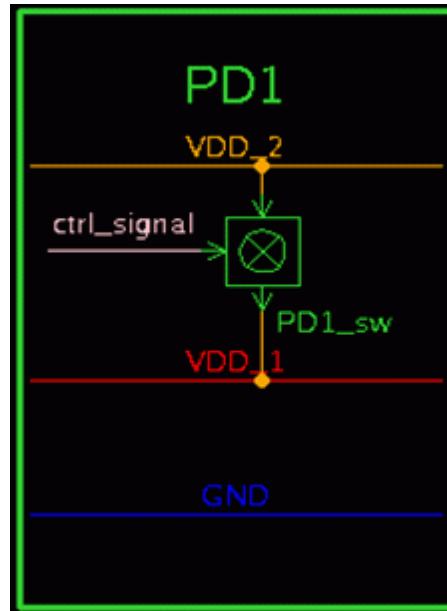
Figure 11-26 Representation of a Power Switch



The symbol indicates the input and output supply ports, the control ports, and the control signals. The arrows represent the direction of the ports.

As shown in [Figure 11-26](#), a power switch can have single or multiple control signals. The power switches are located within the boundaries of their parent power domain. Because power switches have supply nets as input and output, they are located between the power supply nets as shown in [Figure 11-27](#).

Figure 11-27 Location of the Power Switches in the Power Domain



Power State Tables

A power state table defines the legal combination of states that can exist simultaneously during the operation of the design. A power state table is a set of power states of a design in which each power state is represented as an assignment of power states to individual power nets. A power state table of a design captures all the possible operational modes of the design in terms of power supply levels. Given a power state table, a power state relationship (including voltage and relative always-on relations) can be inferred between any two power nets. The power state table is used by the synthesis tool for analysis, synthesis, and optimization of the multivoltage design.

Creating Power State Tables

The `create_pst` command creates a power state table and assigns a name to the table. The command lists the supply ports or supply nets in a particular order. The `add_port_state` defines the name of the possible states for each supply port.

The UPF standard requires a simple name for the *table_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to true.

A supply port and a supply net can have the same name, even when they are unconnected. If such a name is listed in the `create_pst` command, it is assumed to represent the supply port and not the supply net.

The power switch supply ports and internal PG pins of macro cells with fine-grained switch, are considered supply ports because they are connected by supply nets, so they can be listed as supply nets in `create_pst` command.

You can use the component supply nets of a supply set to define a Power State Table. This is because, the state of every component of a supply set can be unambiguously determined, when you define a supply expression for the supply set.

For more details, see the `create_pst` and `add_port_state` command man pages.

Defining the States of Supply Nets

The `add_pst_state` command defines the states of each of the supply nets for one possible state of the design. The command must specify the name of the state, the name of the power state table already created, and the states of the supply ports in the same order as listed in the `create_pst` command.

The listed states must match the supply ports or nets listed in the `create_pst` command in the corresponding order. For a group of supply ports and supply nets directly connected together, the allowable supply states are derived from the shared pool of supply states commonly owned by the members of the group.

The UPF standard requires a simple name for the *state_name* argument. By default, Power Compiler does not check for this requirement. To check that this requirement is met, set the `mv_input_enforce_simple_names` variable to true.

The following example creates a power state table, defines the states of the supply ports, and lists the allowed power states for the design.

```
create_pst pt -supplies { PN1 PN2 SOC/OTC/PN3, FSW/PN4 }
add_port_state PN1 -state { s88 0.88 }
add_port_state PN2 -state { s88 0.88 } -state { s99 0.99 }
add_port_state SOC/OTC/PN3 -state { s88 0.88 } -state { pdown off }
add_port_state FSW/PN4 -state { s0, 0.0 } -state { pdown off }
add_pst_state s1 -pst pt -state { s88 s88 s88 s0 }
add_pst_state s2 -pst pt -state { s88 s88 pdown s0 }
add_pst_state s3 -pst pt -state { s88 s99 pdown s0 }
add_pst_state s4 -pst pt -state { s88 s99 s88 pdown }
```

Note:

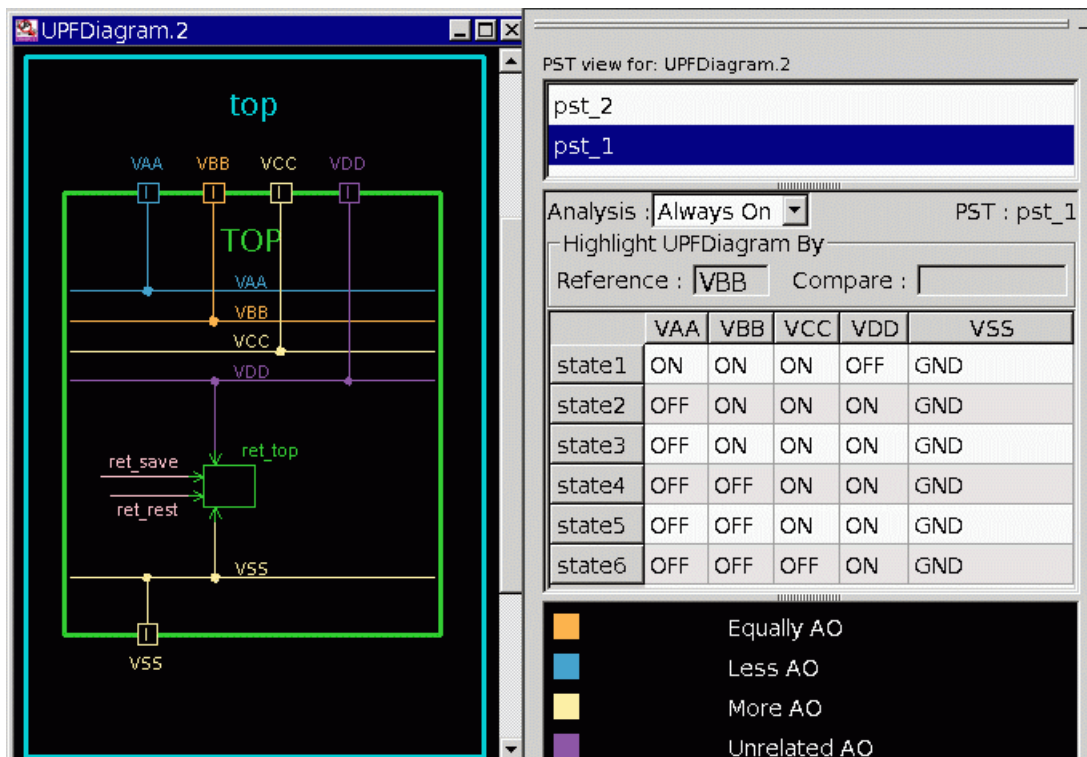
It is an error to define conflicting states on the components of a group of supply sets that you have associated with each other using the `associate_supply_set` command. Also, it is an error to have two conflicting states in the power state table for identical functions of the supply set group that are associated with each other.

Visually Analyzing Power State Tables in the UPF Diagram View

To analyze and debug the isolation and the level-shifter strategies in a UPF design, use the UPF diagram view with the Power State Table panel. You can view the power states for each supply in a power state table and examine their relationships in the UPF diagram.

Figure 11-28 shows an example of the UPF diagram view and Power State Table panel during always-on analysis.

Figure 11-28 Always-On Analysis Using the Power State Table Panel



The Power State Table panel appears automatically when you open the UPF diagram view. You can hide or display this panel by choosing View > Toolbars > Power State Table.

The Power State Table panel provides the following types of analysis:

- *Always-on analysis* compares the on-off states between any two supplies, including both power and ground supplies

- *Multivoltage level-shifter analysis* compares the voltage relationships between supplies

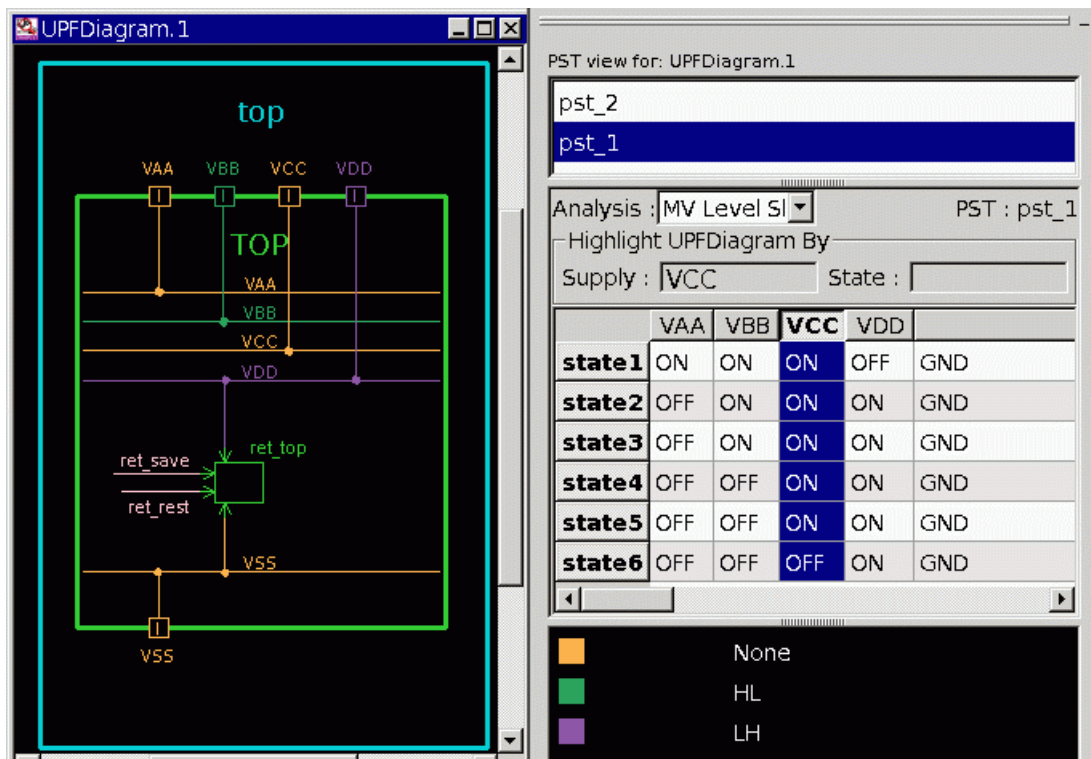
During always-on analysis, you can compare the power and ground supplies in the power state table because the combination of the power and ground supplies defines the always-on relationships between the power domains.

For more details, see, [Analyzing Multivoltage Design Connections in the GUI](#).

During multivoltage level-shifter analysis, to decide if a level shifter is needed between a driver and a load supply, only the power supplies of the power domains need to be compared; the tool supports 0 volts or the off state for the ground supply.

[Figure 11-29](#) shows an example of the UPF diagram view and Power State Table panel during multivoltage level-shifter analysis.

Figure 11-29 Multivoltage Level-Shifter Analysis Using the Power State Table Panel



For more information, see the “Visualizing Power State Tables” topic in Design Vision Help.

Support for Well Bias

Some process technologies allow dedicated voltage supplies, instead of normal rail voltages, to be applied to n-well and p-well regions of the chip. Applying a bias voltage to a well changes the threshold voltage for transistors in the well, affecting the performance and leakage current.

The Power Compiler tool offers an optional mode to specify the n-well and p-well bias supply infrastructure using UPF commands. In this mode, the tool automatically makes supply connections to the well bias pins.

To enable the UPF-based well bias mode, set the `enable_bias` design attribute to `true` in the UPF command file as follows:

```
set_design_attributes -elements {.} -attribute enable_bias true
```

With the `enable_bias` design attribute set to `true`, supply sets and supply set handles can be used to specify the bias supply connections. The well bias pins, along with the power supply and ground pins, are connected automatically for standard cells, macros, and other types of cells.

For more information about using this mode, see “N-Well and P-Well Bias” in the *Synopsys Multivoltage Flow User Guide*.

For more information about well bias modeling, see the *Library Compiler User Guide*.

Inserting Power Management Cells

Power management cells such as level shifters and isolation cells are not usually part of the original design description. They are inserted during the logic synthesis flow. Buffer-type level shifters can be inserted by the tool as part of compilation or manually by instantiating the cells in RTL, or using specific commands that insert level shifters. Similarly isolation cells and enable-type level shifters can be instantiated at the RTL level of the design description or inserted by using commands that insert isolation cells.

You can also insert these cells by using the `insert_mv_cells` command. This command uses the strategies defined in the UPF file, when inserting these cells. Using the various options of the `insert_mv_cells` command, you can choose to insert only the isolation cells or only the level shifter cells, or both. By default, the command inserts both isolation and level-shifter cells. You can use this command on both RTL and gate-level designs.

When inserting isolation and enable level-shifter cells, the naming convention of the inserted cells is specified as follows:

```
<name_prefix>_snps_<power_domain_name>_<isolation_strategy_name>_snps_  
<pin_name>_<instance_index>_<name_suffix>
```

For enable level-shifter cells with both a related isolation strategy and a level-shifter strategy, the isolation and level-shifter prefix and suffix are added to the new name. The new naming convention for these enable level-shifter cells is as follows:

```
<levelshift_prefix><isolation_prefix>_snps_<power_domain_name>_<isolation_strategy_name>_snps_<pin_name>_<instance_index>_<isolation_suffix><levelshift_suffix>
```

There is no change in the naming of level-shifter and retention cell instances.

The `insert_mv_cells` command inserts the power management cells in the following order:

1. Repeaters or buffers
2. Isolation cells
3. Level-shifter cells
4. Enable level-shifter cells. Based on the requirement, replace the isolation cells by enable level-shifter cells.

[Table 11-6](#) summarizes the command option and command sequences that can result in the insertion of enable level-shifter cells.

Table 11-6 *Command Sequences and Enable Level-Shifter Cell Insertion*

Command option and sequence	Enable level-shifter cell inserted?
<code>insert_mv_cells -all</code>	yes
<code>insert_mv_cells -isolation -level_shifter</code>	yes
<code>insert_mv_cells -isolation</code> <code>insert_mv_cells -level_shifter</code>	yes
<code>insert_mv_cells -level_shifter</code> <code>insert_mv_cells -isolation</code> <code>insert_mv_cells -level_shifter</code>	yes
<code>insert_mv_cells -level_shifter</code>	no
<code>insert_mv_cells -isolation</code>	no
<code>insert_mv_cells -level_shifter</code> <code>insert_mv_cells -isolation</code>	no

Note:

You must uniquify your design by using the `uniquify` command before inserting the power management cells. Otherwise, Power Compiler issues an error message.

Reviewing the UPF Specifications

After specifying the power constraints using UPF, you can review the design using the commands or using the GUI.

Commands to Query and Edit Design Objects

To query and edit design objects, use the following commands:

- `find_objects`

The command finds logical hierarchy objects within the specified scope and returns the hierarchical names that match the specified criteria. The command returns a null string when nothing matches the specified search pattern.

- Query commands

To query the UPF objects, use the following commands. The query is resolved when the command is executed, and the result of the query is used by Power Compiler.

- `query_cell_instances`

Returns a list of instance names for all instances of a given reference cell in the current scope of the design.

- `query_cell_mapped`

Returns the reference cell name of a given cell instance.

- `query_net_ports`

Returns a list of ports logically connected to a specified net. By default, the command returns only the ports present at the level of the current scope.

- `query_port_net`

Returns the name of the net logically connected to a specified port, if any such net exists.

- `query_port_state`

Returns information about the port states that have been previously defined using the `add_port_state` command

- `query_pst`

Returns information about the power state tables previously defined with the `create_pst` command

- `query_pst_state`
Returns information about the states that have been previously defined with the `add_pst_state` command
- `query_power_switch`
Returns information about the power switches previously defined with the `create_power_switch` command
- `query_map_power_switch`
Returns information about the power switch library cells previously mapped to the UPF power switches with the `map_power_switch` command
- Editing commands
The editing commands are not written in the UPF file written by the `save_upf` command. However, the changes to the netlist are available in the Verilog and VHDL netlist written by Power Compiler.
 - `connect_logic_net`
 - `create_logic_net`
 - `create_logic_port`

For more details, see the command man pages.

Reviewing the Power Intent Using the Design Vision GUI

The Power menu in the GUI allows you to specify, modify, and review your power architecture. It also lets you view the UPF diagram and examine the UPF specification defined in your design.

If you have not defined the power intent for your design, see [Defining Power Intent in UPF](#).

If you have already defined the power intent for your design, the Visual UPF dialog box displays the details of your power specification. Using the Power Domains and Power Domain Properties sections, you can edit the power definitions: add new components, redefine the association of the hierarchical cells with the power domains, delete a power domain, and so on.

To open the Visual UPF dialog box,

- Choose Power > Visual UPF

When you open the Visual UPF dialog box, the Visual UPF appears, as shown in the example in [Figure 11-3](#).

The Visual UPF views that show your power intent are

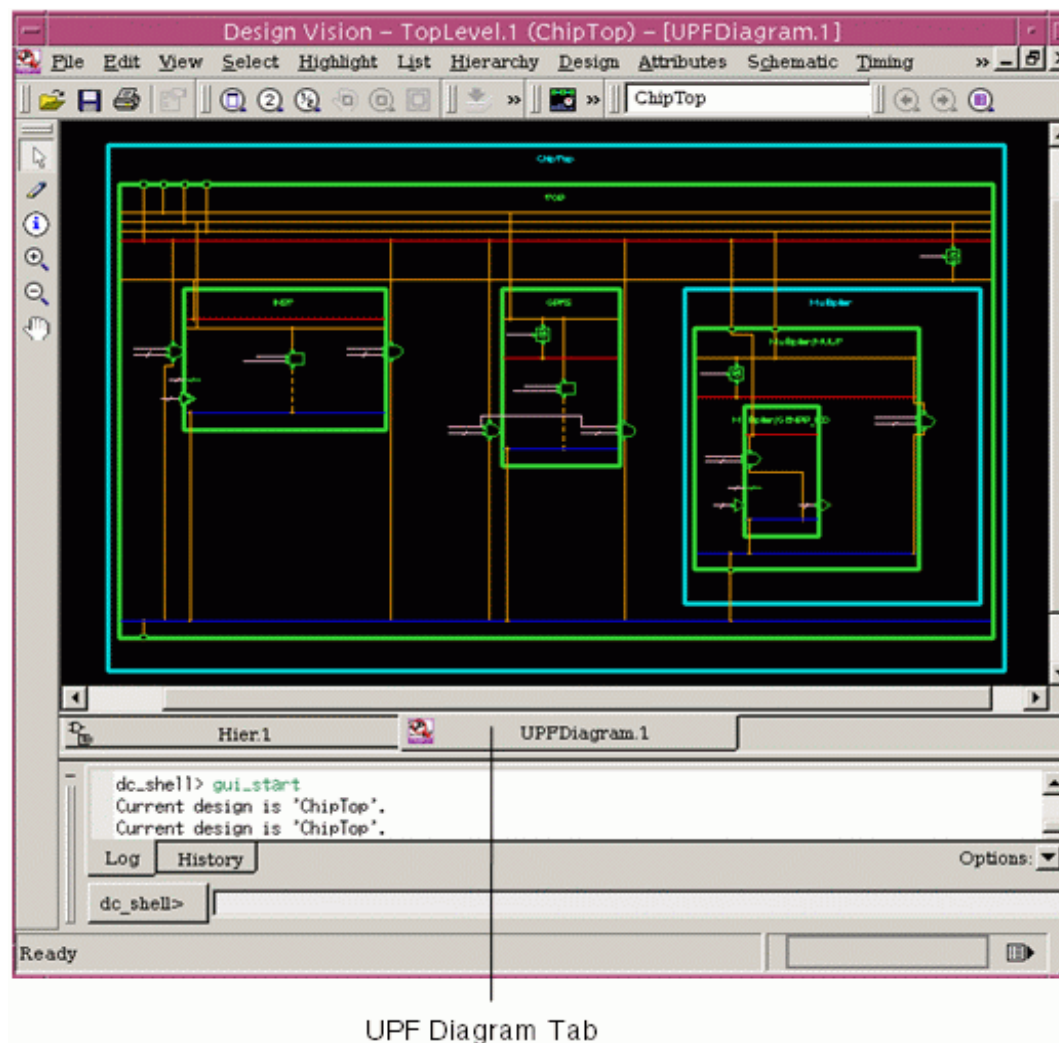
- Design or Logic Hierarchy View

Select the Design Hierarchy tab to view the logic hierarchy of your design, as shown in [Figure 11-3](#).

- UPF Diagram view

Select the UPF Diagram tab to view the pictorial representation of your power definitions as shown in [Figure 11-30](#).

Figure 11-30 UPF Diagram View in the Visual UPF Dialog Box

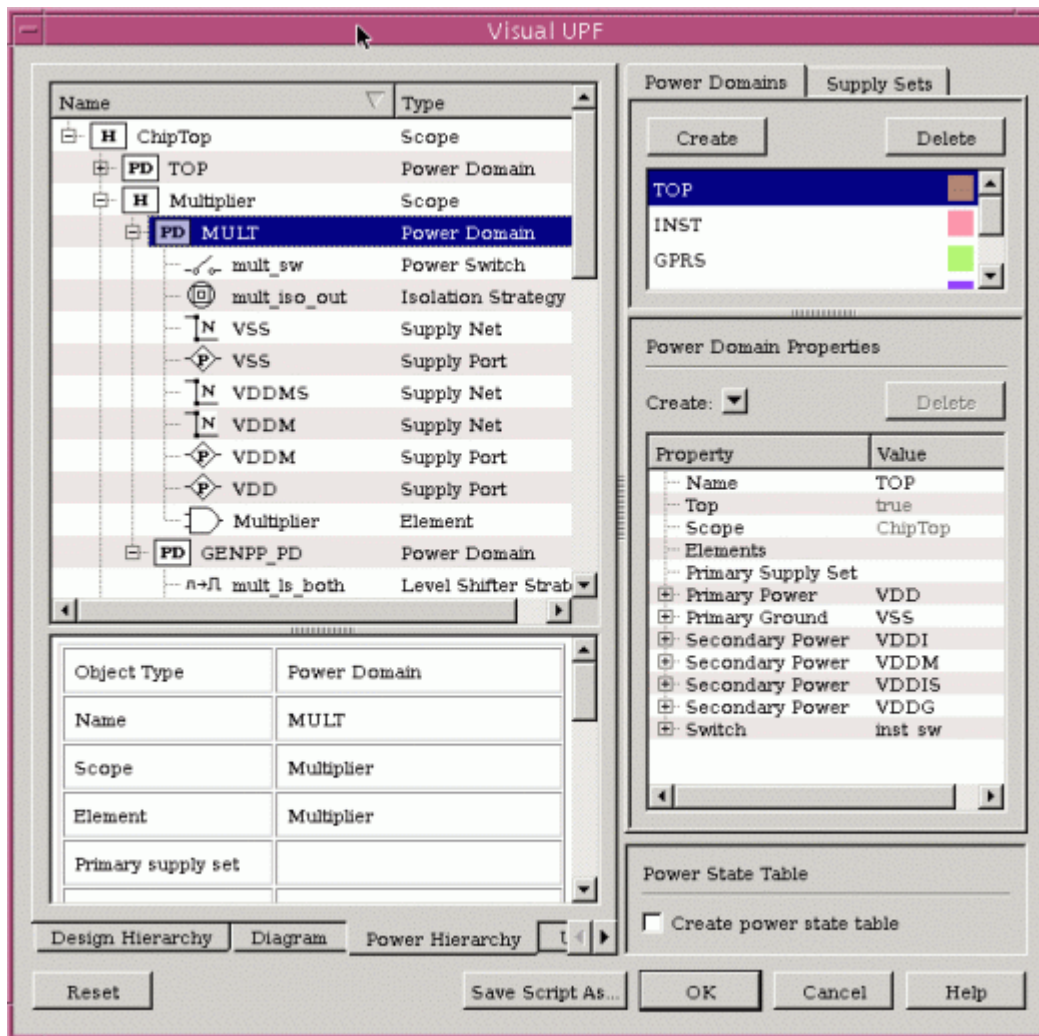


- Power Hierarchy view

Select the Power Hierarchy tab to see the power hierarchy of your design. [Figure 11-31](#) shows the Power Hierarchy view of a design.

The Power Hierarchy view has two sections. The section on the top shows the hierarchy tree with the connections between different power objects. The section at the bottom shows more details and properties of the object that you select in the top section.

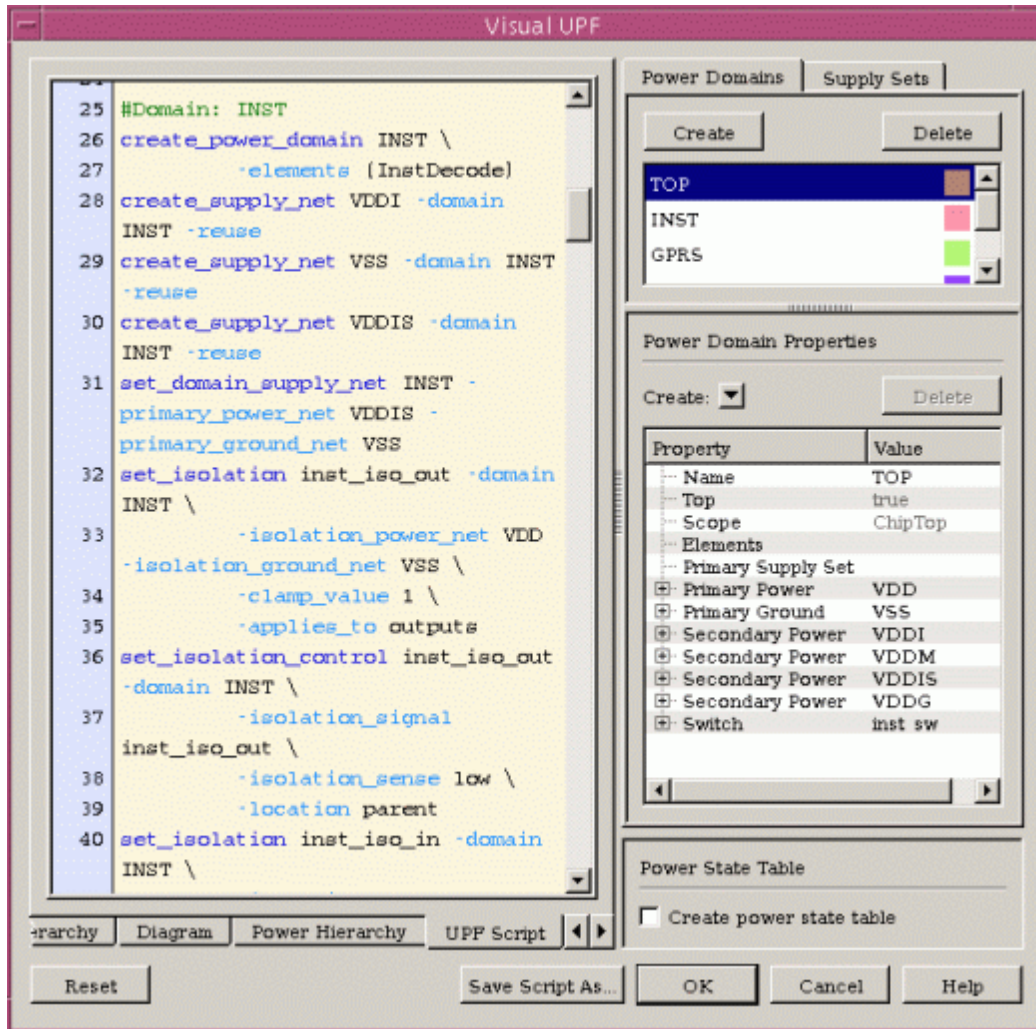
Figure 11-31 Power Hierarchy View in the Visual UPF Dialog Box



- UPF Script view

Use the UPF script tab to view the UPF script for your power definitions. [Figure 11-32](#) shows the UPF Script view. The various colors used in the script help in differentiating the UPF commands and the power objects.

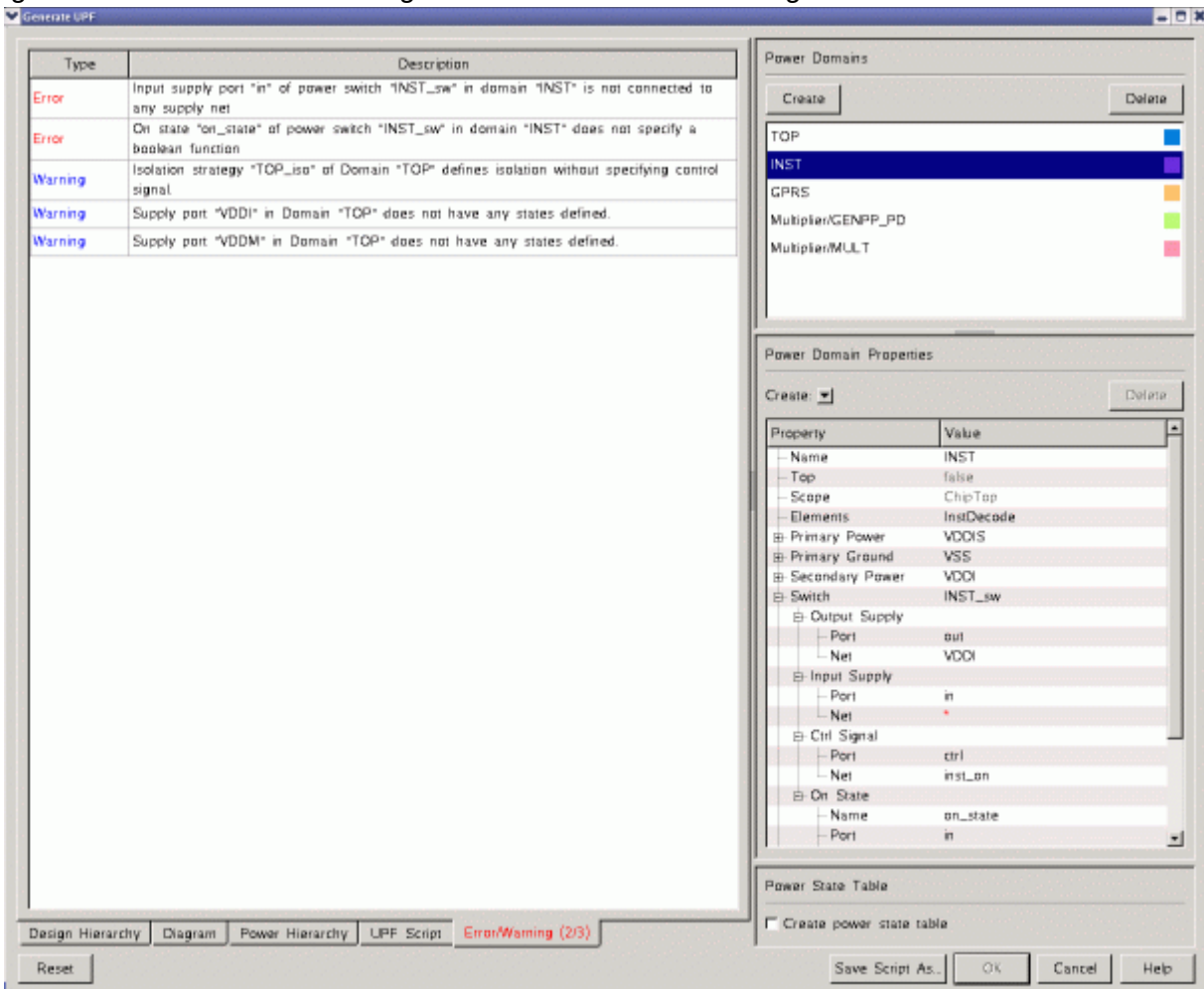
Figure 11-32 UPF Script View of the Visual UPF



- Error and Warning view

The Error/Warning tab in the Visual UPF view becomes active when your modifications cause errors or warnings as shown in [Figure 11-33](#). When there are no errors or warnings, this tab is disabled. You can see the details of the error and warning messages in this view.

Figure 11-33 Error and Warning View in the Visual UPF Dialog Box



Applying the Power Intent Changes

After defining or modifying the power intent, you can do one of the following:

- Save the power intent as a UPF script

Click the Save Script As button to save the power intent script in a file. The file is saved in ASCII format, as a UPF file, but the power intent is not applied to the design database

of the tool. You can run this script either in the batch mode or interactively, to apply the power intent.

This feature can be useful when your changes are not yet complete, and you have to save it for a later use. It can also be useful when you have to edit the file before running it. For example, when you create a power state table, all the possible power states are populated in the table. Before running the script, you must edit the script to remove or comment the states that are not required.

- Apply the power intent to the design database

Click OK to apply the power intent in the Power Compiler design database. Until you click OK, power intent specifications are only contained in the Visual UPF dialog box and do not affect the design database.

Examining and Debugging UPF Specifications

To perform multivoltage-specific checks at various stages of synthesis of your multivoltage designs, use the `check_mv_design` and the `analyze_mv_design` commands:

In addition, the `check_library` command in Library Compiler supports specific checks that are useful in the UPF Flow. For more details, see the Library Checking chapter in the *Library Quality Assurance System User Guide*.

The `check_mv_design` Command

To check for design errors, including multivoltage constraint violations, electrical isolation violations, connection rules violations, and operating condition mismatches, use the `check_mv_design` command.

MV Advisor GUI

The MV Advisor violation browser provides a visual analysis and debugging environment for multivoltage design violations. You can check the design for issues such as multivoltage constraint violations, electrical isolation violations, connection rule violations, and operating condition mismatches. After checking a design, you can use the violation browser to examine the violation report.

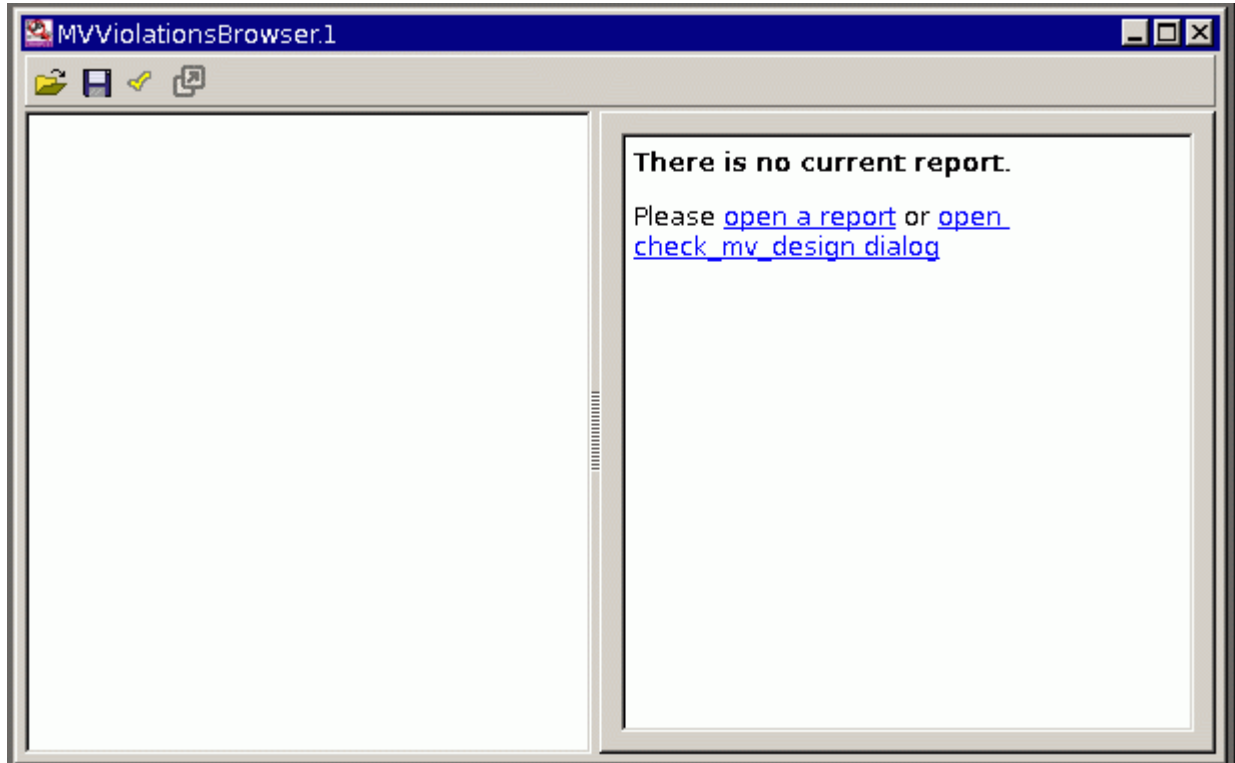
To open the MV Advisor violation browser,

- Choose Power > MV Advisor.

The violation browser automatically loads the current violation report if a valid report is available for the current state of the design. If a valid report does not exist, the violation

browser provides links that you can use to load a saved report or generate a new report, as shown in [Figure 11-34](#).

Figure 11-34 Violation Report When a Valid Report is Not Available



The violation browser groups the violations based on specific properties, displays detailed information about the violations, and guidance for investigating and fixing them. When you select a violation, the violation browser displays details such as an explanation of the warning or error message and suggestions for fixing the violation.

The violation browser also provides access to context-aware reports and other analysis tools. You can

- Select pin names and view information about the pins
- Display man pages (in the man page viewer) for warning and error messages
- Visually inspect a violation by displaying it in a schematic view

You can display the report for an individual violation in a new instance of the Design Vision window that serves as a debugging work environment.

You can check the design for violations before or after you open the violation browser. To check the design before opening the violation browser, use the `check_mv_design` command. When the violation browser is open, you can use the Check MV Design dialog box to check the design.

Checking for Design Violations

You can analyze multivoltage design problems by checking the design for errors and generating a violation report that you can view in the console log view and save in a file. You can check the design at any time by using the `check_mv_design` command. For more information, see the man page. When the MV Advisor violation browser is open, you can check the design by using the Check MV Design dialog box.

When you check the design, the tool creates or updates the current violation report by default. If you do not specify a file name, the tool stores the current report information in a temporary file until the end of the current session. If you specify a file name, the tool saves the report in an XML file and also creates an XSLT file with the same name with a `.xslt` extension. The XSLT file contains auxiliary information that is required when you view the violation report in a Web browser.

The MV Advisor violation browser supports the violations listed in [Table 11-7](#). The `check_mv_design -output` command identifies and reports several other issues and these are not supported by the GUI for reporting and fixing details.

Table 11-7 Violations Supported by MV Advisor

LIBSETUP-001	LIBSETUP-001a	LIBSETUP-001b	MV-038	MV-044	MV-076	MV-078
MV-166	MV-168	MV-168b	MV-231	MV-231a	MV-231b	MV-232c
MV-232l	MV-237	MV-252	MV-513	MV-514	MV-514a	MV-514b
MV-516	MV-529	MV-534	MV-545	UPF-067	UPF-103	

Violation Groups

The violation browser groups the messages based on the source domain-sink domain pair in the first level and the driver pin in the next level, as shown in [Table 11-8](#).

Table 11-8 Messages Grouped by Source Domain and Sink Domain Pair

Message ID	Description
MV-231	Level-shifter violations
MV-231a	
MV-231b	

Table 11-8 Messages Grouped by Source Domain and Sink Domain Pair (Continued)

Message ID	Description
MV-237	Paths with voltage violations
MV-252	Missing level-shifting violations specified with the <code>-no_shift</code> option
MV-513	Redundant isolation
MV-514 MV-514a MV-514b	Missing isolation violations
MV-545	Missing isolation violations with the <code>-no_isolation</code> option

The violation browser groups the messages shown in [Table 11-9](#) based on the power domains where the cells or nets are located.

Table 11-9 Messages Grouped by Power Domain of the Violating Cells or Nets

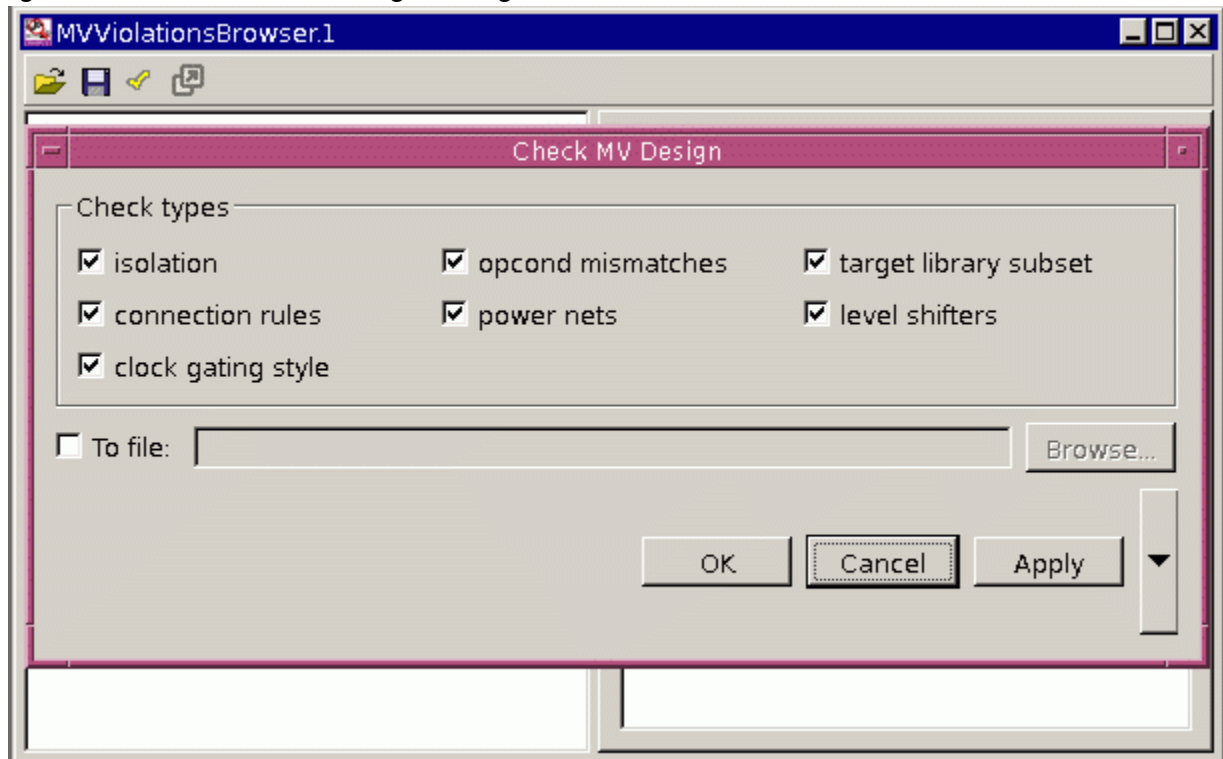
Message ID	Description
LIBSETUP-001 LIBSETUP-001a LIBSETUP-001b	Cells with mismatched operating conditions
MV-044	Isolation cells used as a core cell
MV-076	Always-on nets driven by a normal cell
MV-078	Always-on cells driving a normal net
MV-166	Retention cells without a strategy
MV-168, MV-168b	Isolation cells without a strategy
MV-232c MV-232l	Level-shifter with violations
MV-516	Back-to-back isolation cells violation
MV-529	Unused power management cells
UPF-067	Undetermined PG pin connection

In addition, the violation browser groups

- MV-038 warning messages under one title and without multiple levels of groups
- MV-534 messages based on the driver pins
- MV-232c messages based on the power domains in the first level and the power supplies in the next level
- MV-516 messages under one title and without multiple levels of groups
- UPF-103 messages based on the ignored strategy

To generate or update the current violation report,

1. Click the  button to run the `check_mv_design` command and load the report in the MV Advisor violation browser. The Check MV Design dialog box appears as shown in [Figure 11-35](#).
2. Select or deselect the check type options as needed, to perform the required checks. The tool performs all the checks by default.
3. Select the “To file” option.
4. (Optional) Specify a file name, to save the report in a file.
5. Click OK or Apply.

Figure 11-35 Check MV Design Dialog Box

The Web browser report groups the violations in the same way that the MV Advisor violation browser groups them. However, the Web browser displays only the violation list, the information in the violation category tree in the violation browser. It does not display the detailed report information for each violation that the violation browser displays.

To open a violation report in your Web browser,

- Open the XML file in a Web browser window.

The XSLT file must be present in the same directory.

Examining Design Violations in the MV Advisor Violation Browser

The MV Advisor violation browser provides a visual analysis and debugging environment for design violations in a multivoltage design. You can search for and view information about several types of multivoltage design violations. When you select a violation, the violation browser displays details about the violation and guidance about how to proceed in investigating the violation.

The violation browser view window consists of a button bar at the top and two panes: a violation category tree on the left and a report view on the right.

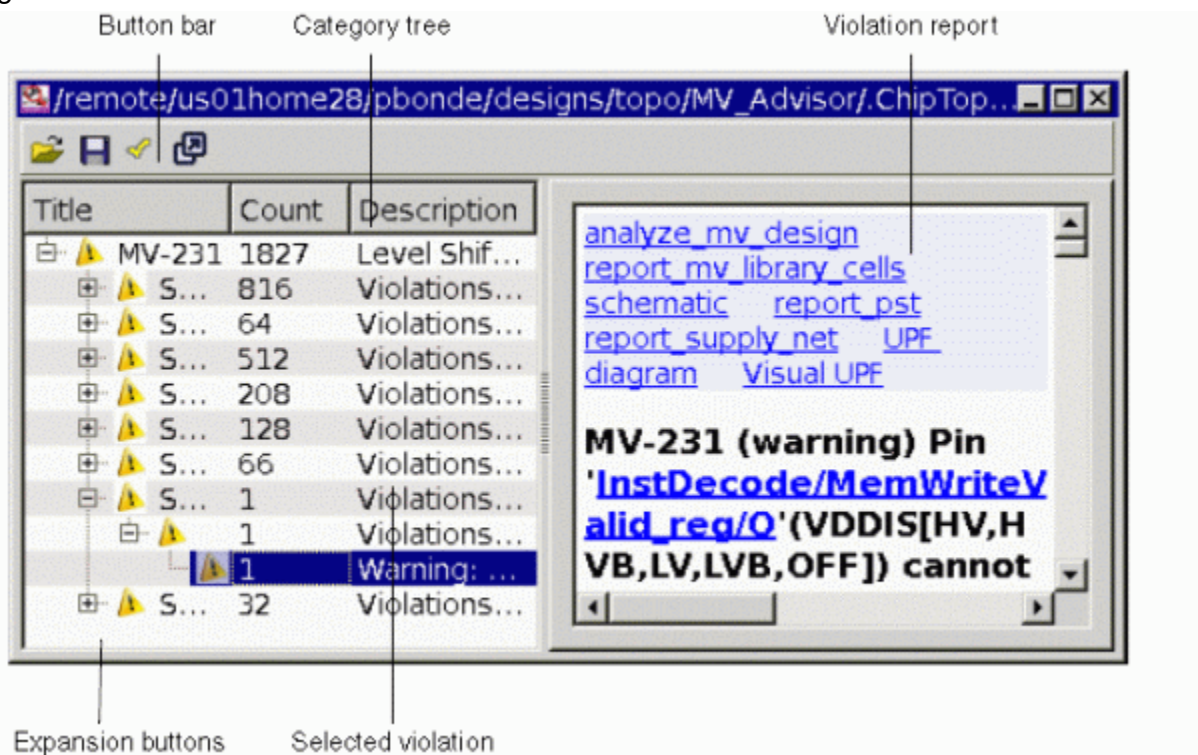
- The violation category tree groups the violations into types, categories, and subcategories.

You can use the expansion buttons in the category tree to expand or collapse individual types, categories, and subcategories.

- The report view displays information about the type, category, or violation that you select in the category tree.

Figure 11-36 shows an example of the MV Advisor Violation Browser with a violation selected in the category tree and the violation report in the report view.

Figure 11-36 The MV Advisor Violation Browser



The buttons on the button bar at the top of the view window allow you to

- Open another violation report
- Save the violation report you are viewing
- Generate or update the current violation report
- Display the report for the selected violation in a new Design Vision window, where you can use other analysis tools to debug the violation

To help you to evaluate the status of violations through the design flow or to compare the reports from different design checks, you can open more than one report at the same time. When you open a report file, the violation browser compares the design name and the number of cells with the netlist in the current design, and displays a message if it finds any inconsistencies.

Note:

The Design Vision GUI supports detailed reporting of how to debug and fix violations for a subset of the issues identified and reported by the `check_mv_design -output` command. For information about the checks supported in the MV Advisor violation browser, see [Checking for Design Violations](#).

Exploring the Violations

To analyze and debug a violation, you can open the report on the Browser panel in a new Design Vision window. By default, the Browser panel is attached to the left side of the window. You can use the workspace area in this window to debug the violation by clicking links at the top of the violation report to generate and display other reports and open analysis views such as a schematic or UPF diagram view.

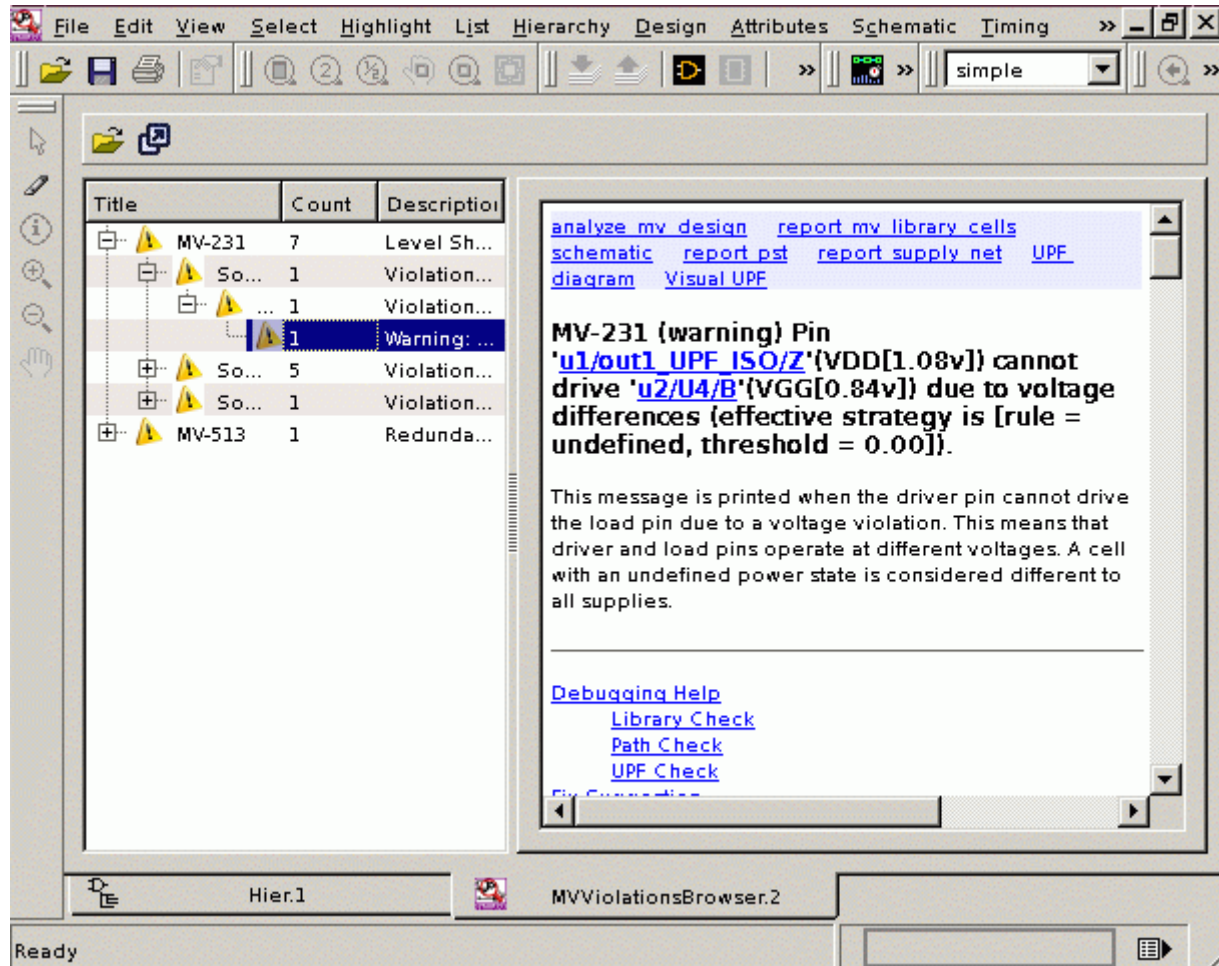
To explore the violation categories and view the violations within each category, do the following:

1. Expand violation categories, showing the subcategories or violations at the next level in the category tree.

To expand or collapse a violation category, double-click its line in the category tree or click its expansion button in the Type column.

As shown in [Figure 11-37](#), a plus sign on the expansion button means the category is collapsed. A minus sign on the expansion button means the category is expanded.

Figure 11-37 Detailed Report in the Report View



2. Select a violation type, category, or subcategory tree to display information about it in the report view.

When you select a violation type, the report view displays the generic violation message and the number of violations of that type found in the design as shown in [Figure 11-37](#).

When you select a violation category or subcategory, the report view displays the generic violation message, the number of violations in the category, and the location of the violations.

3. Select a violation in the category tree to display a detailed report about it in report view, as shown in [Figure 11-37](#).

When you select a violation, the report view displays the warning or error message, a brief explanation of the message, and a detailed description of the violation that includes debugging information and suggestions for fixing the violation.

The links at the top of the report view allow you to run various report commands and other features of the tool.

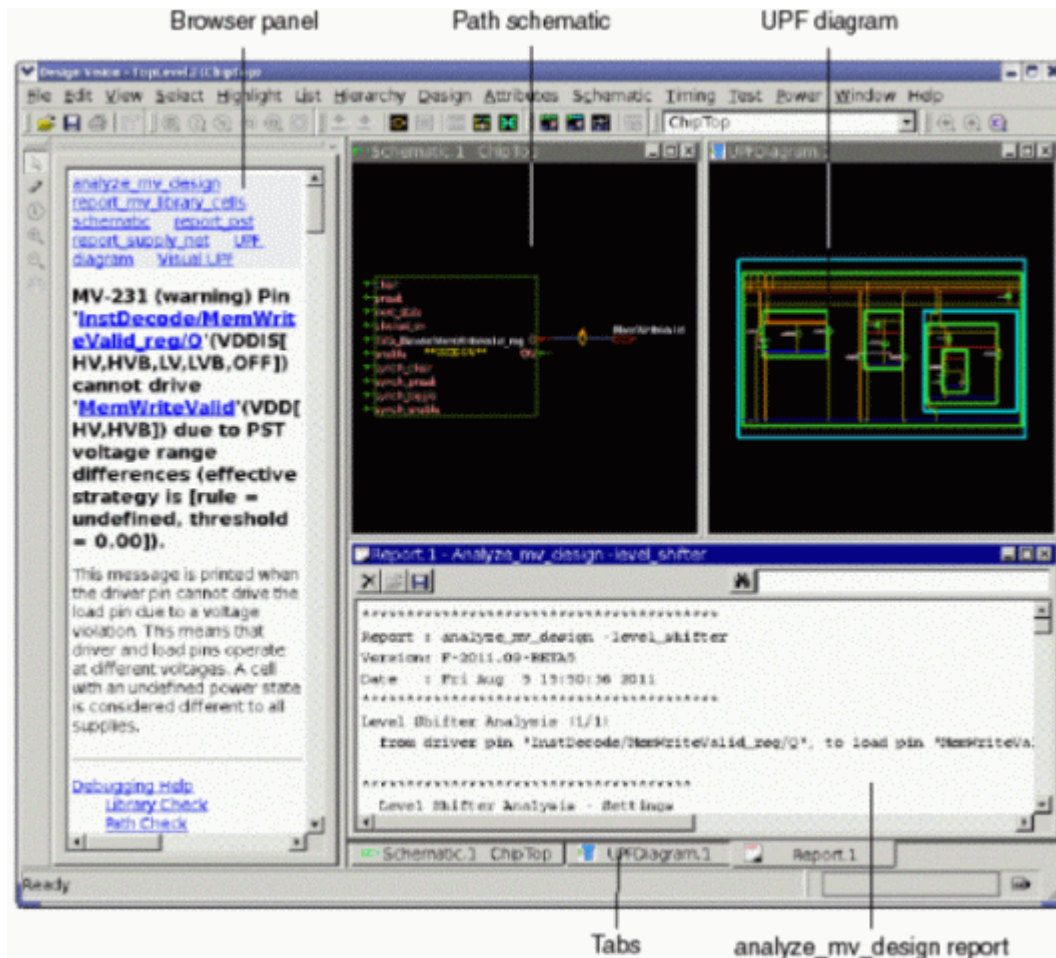
- To run the `analyze_mv_design` command and display the level shifter report in a report view, click the `analyze_mv_design` link
- To run the `report_mv_library_cells` command and display the library cells report in a report view, click the `report_mv_library_cells` link
- To run the `report_pst` command and display the power state table report in a report view, click the `report_pst` link
- To run the `report_supply_net` command and display the supply net report in a report view, click the `report_supply_net` link
- To view the violation in a schematic, click the schematic link
- To open a UPF diagram view, click the UPF Diagram link
- To open the Visual UPF dialog box, click the Visual UPF link

In the warning or error message, click the pin name to select the pin. In the detailed report, you can click the links to run commands and access useful features of the tool.

You can use the arrow keys on the keyboard to scroll through the report view. To scroll up or down, press the Up Arrow key or the Down Arrow key. To move to the top or bottom of the view, press the Page Up key or the Page Down key.

Figure 11-38 shows an example of the debugging environment provided by the Browser panel in a new Design Vision window. For example, after displaying the violation report on the Browser panel, you can click links at the top of the report to display the violation in a schematic, open the UPF diagram view, and see the reports in report views.

Figure 11-38 Violation Report on Browser Panel in a New Window

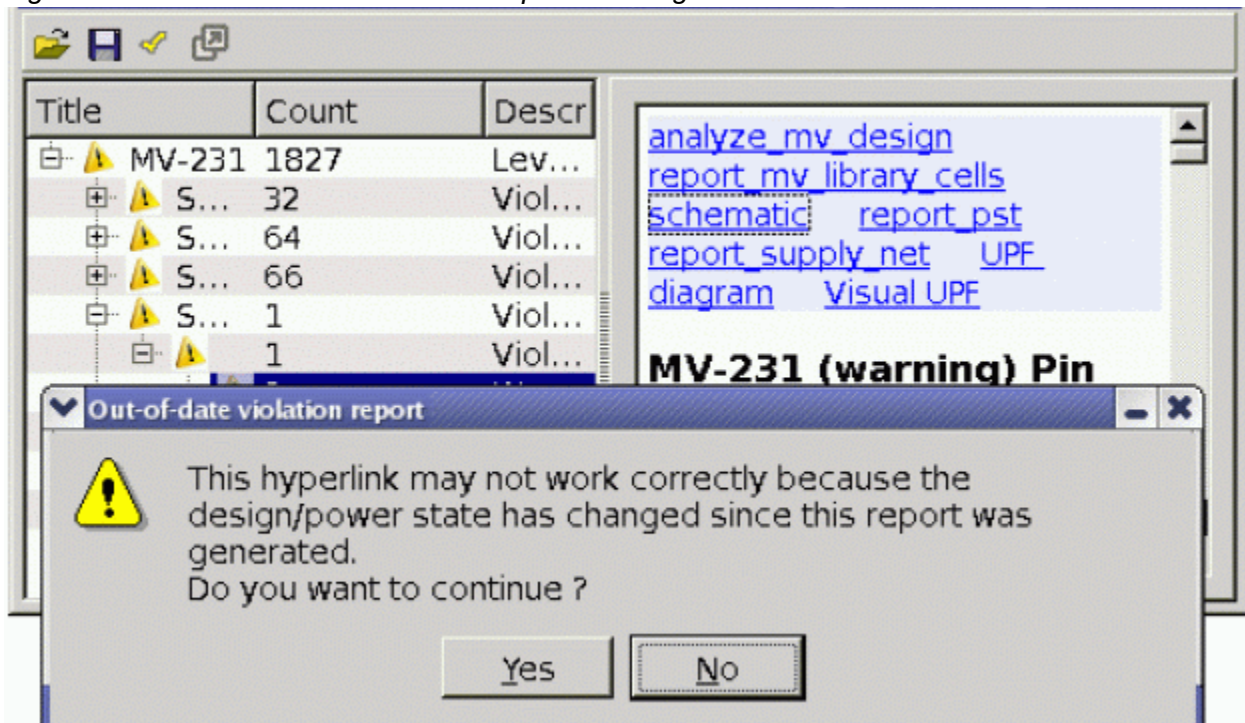


The tool maintains a single, current repository for the multivoltage design violations that you can view in the MV Advisor violation browser. If you update the violation report that you are viewing, the tool automatically refreshes the report information in the violation browser.

If you view an out-of-date report in the violation browser, or if a change that you make in the design invalidates the report that you have open in a violation browser window, the term “Out-of-date” appears in the window title bar. In addition, the violation browser restricts hyperlinks in an out-of-date report when a link action might update the design or manipulate design objects. If you click a restricted hyperlink, the tool displays a warning and prompts you to continue or cancel the link action, as shown in [Figure 11-39](#).

Reports, schematics, the UPF diagram, and the Visual UPF dialog box all work with the up-to-date design. Commands that operate directly on an element reported in a violation, such as the `report_net` command, can cause an error if the element no longer exists in the design due to a previous action. Hyperlinks that are not restricted include internal HTML jumps, man page links, and links to preview commands.

Figure 11-39 Out-Of-Date Violation Report Warning



The analyze_mv_design Command

The `analyze_mv_design` command reports path-based design details of a multivoltage design that can be useful in debugging multivoltage design issues. The report contains details of the variable settings for level-shifter insertion and always-on buffering, relevant power state tables, the driver-to-load pin connections, the pin-to-pin information on specified paths, the target libraries used for insertion of power management cells, and other useful debugging information. You can also run this command in the GUI and see the issues identified, in the schematic. For more details, see [Analyzing Multivoltage Design Connections in the GUI](#).

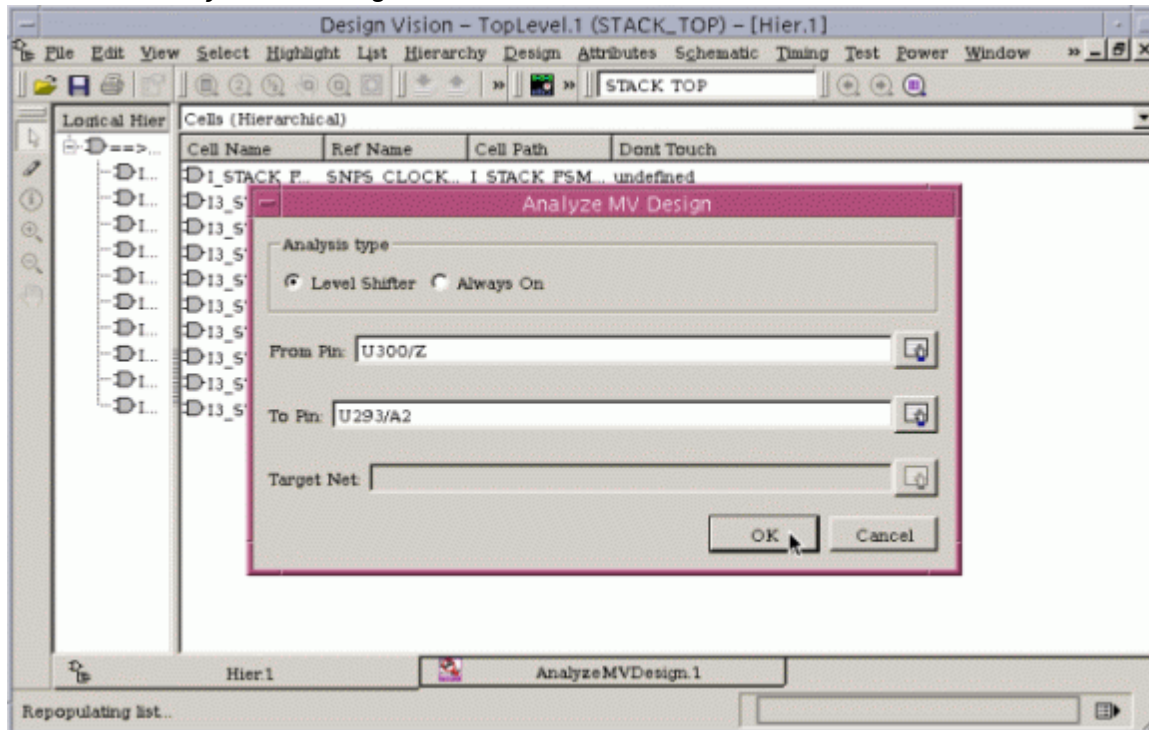
For more information, see the `analyze_mv_design` command man page.

Analyzing Multivoltage Design Connections in the GUI

You can use the Analyze MV Design dialog box to analyze your design for multivoltage-specific connectivity issues. The `analyze_mv_design` command runs internally and displays the result in a new view.

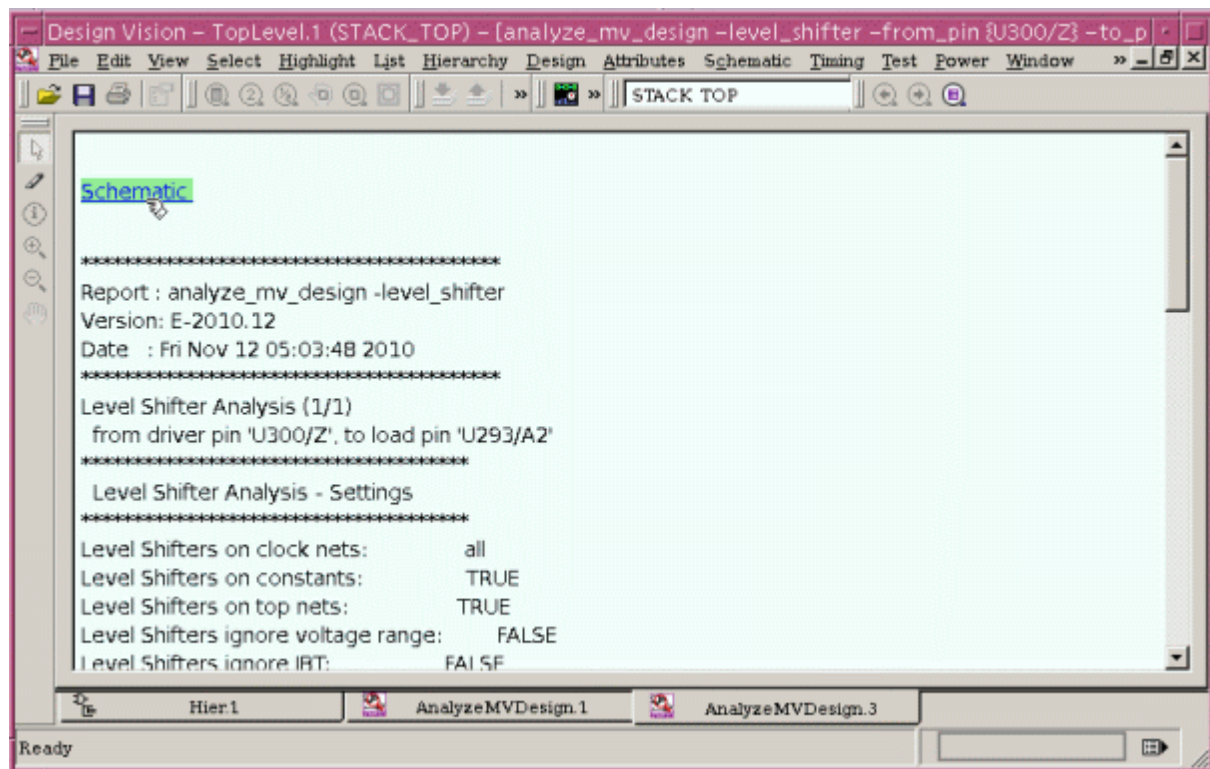
To open the Analyze MV Design view, choose **Power > Debugging > Analyze MV Design**. The Analyze MV Design dialog box appears as shown in [Figure 11-40](#).

Figure 11-40 Analyze MV Design Window



Use the dialog box to choose the type of analysis to perform, either level shifter or always on. You can also specify the From Pin and the To Pin, where the checks have to be performed. When you click OK, the tool runs the `analyze_mv_design` command.

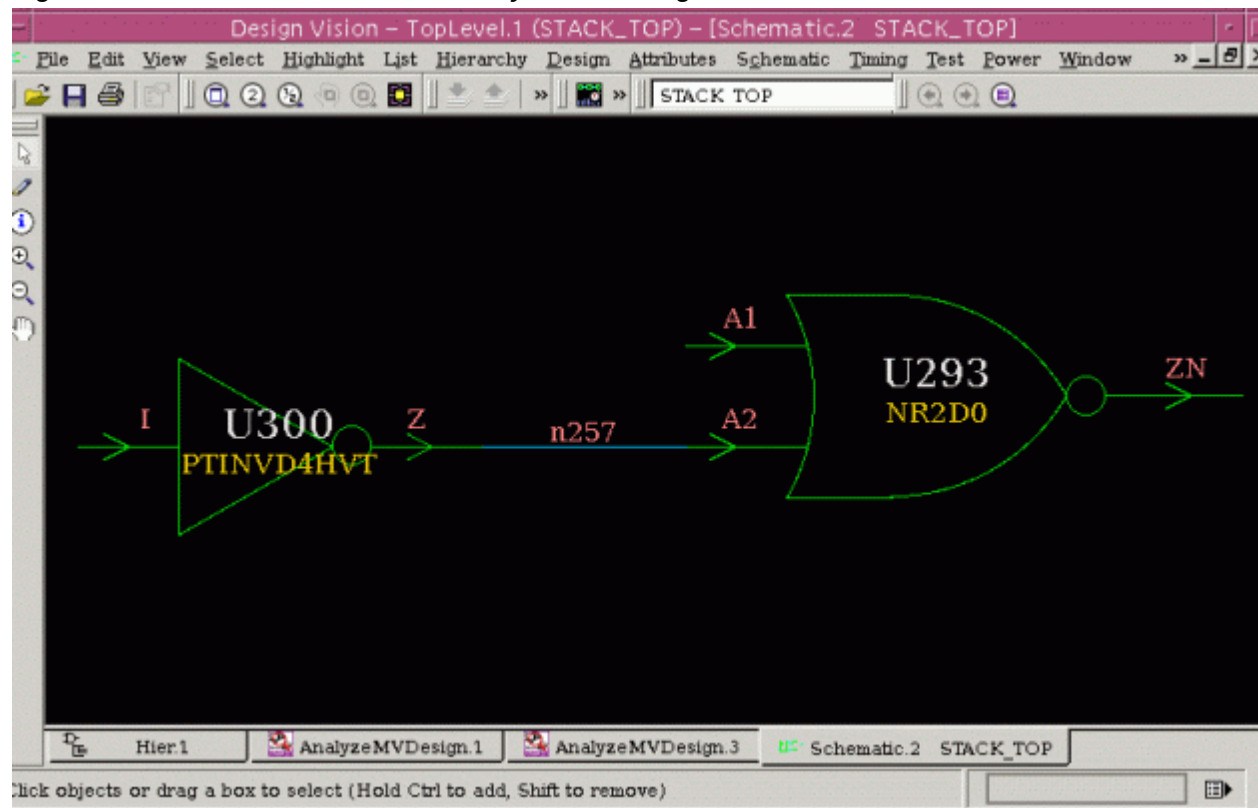
The report of the `analyze_mv_design` command is displayed in a new view, as shown in [Figure 11-41](#). The report contains details of level-shifter violations.

Figure 11-41 Report View of the Analyze MV Design Window

The report also contains a hyperlink to the schematic; when you follow the link, the schematic shows design objects that are specific to the reported issue, as shown in [Figure 11-42](#). In the schematic, you can

- Create a collection of the power supply nets connected to one or more pins
- View a list of the ground supply net connections for one or more pins
- View a report of power pin information for one or more cells
- View a report of PG pin library information for one or more cells

Figure 11-42 Schematic View of Analyze MV Design Window



Writing the Power Information

The power information updated by Power Compiler during synthesis can be written out with the `save_upf` command. This UPF file written by Design Compiler is referred to as the UPF' file to distinguish it from the UPF file that you use to synthesize the design. The UPF' file is used as input to the downstream tools, such as IC Compiler, PrimeTime, PrimeTime PX, and Formality.

The additional information in the UPF' file are

- A comment on the first line, as shown in the following example:

```
#Generated by Design Compiler(E-2010.12) on Thu Oct 28 14:26:58 2010
```
- Explicit power connections to special cells such as level shifters and dual supply cells

- Additional UPF commands specified at the command prompt in the Design Compiler session

If you specify UPF commands at the command prompt, along with the RTL file, update the UPF file with these commands. This update is required for Formality to verify the design successfully.

The UPF standard requires a simple name for the argument of certain UPF commands. By default, Power Compiler does not check for this requirement. To check that this requirement is met in the UPF file written by the `save_upf` command, set the `mv_output_enforce_simple_names` variable to `true`.

Preserving the Command Order in the UPF' File

To improve the clarity of the UPF' file, the tool

- Writes the user-specified UPF commands and tool inserted UPF commands in separate sections
- Lists the commands in the user-specified section in the order they were specified

To distinguish the user-specified UPF command section from the tool-generated UPF command section, the sections are separated by the `derived_upf` variable setting.

The beginning of the tool-generated section is marked by the following setting:

```
set derived_upf true
#Design Compiler added commands
```

The end of the tool-generated section is marked by the following variable setting:

```
set derived_upf false
```

Do not explicitly set the `derived_upf` variable to either `true` or `false`. Use of this variable is restricted to the tool.

If the RTL design contains PG information, the UPF commands generated by the `convert_pg` command are listed in a separate section, marked by the following comments:

```
# Commands created by "convert_pg"
```

```
# End of commands created by "convert_pg"
```

When a UPF' file is read into Power Compiler and another UPF' file is written, the command order is preserved in the newer UPF' file. The file contains the user-specified UPF command and the tool generated UPF commands in separate sections.

Note:

This feature is not supported in a hierarchical flow. If you use this feature in the hierarchical flow, Power Compiler issues the UPF-401 information message.

[Example 11-6](#) shows the UPF' file written out by Power Compiler.

Example 11-6 UPF' File Generated by Power Compiler

```
#Generated by Design Compiler

create_power_domain PDT
create_supply_net SN1 -domain PDT
create_supply_net SN2 -domain PDT

create_power_domain PDC -elements {ABC}
create_supply_net SN3 -domain PDC

set derived_upf true
#Design Compiler added commands
connect_supply_net SN1 -ports {PORT1}
connect_supply_net SN3 -ports {PORT2}
set derived_upf false
```

Controlling the Line Width in the UPF' File

When the `save_upf` command writes the commands into an output file, by default, commands with several arguments are not split into multiple lines. To simplify the format of the output file for ease of use, you can control the line width of the file by using the following variables:

- `mv_output_upf_line_width`

Use this variable to control the line width in the UPF file written by the `save_upf` command. The default of this variable is 0, indicating that the `save_upf` command does not split long lines over multiple lines.

Set this variable to a positive integer to specify the width of each line. Lines that are longer than the specified value are split and written on multiple lines. When a line is split over multiple lines, the end of each incomplete line that continues on the next line is marked with a backslash character (`\`), as shown in [Example 11-7](#).

When the line to be split does not have a space up to the specified limit, the tool allows the line to exceed the limit; the line is split at the first space after the specified limit.

- `mv_output_upf_line_indent`

Use this variable to specify the number of spaces to indent at the beginning of each line continuation. The default of this variable is 2 which means, the `save_upf` command indents 2 spaces at the beginning of each continuing line.

[Example 11-7](#) shows the UPF file written by the `save_upf` command, when the `mv_output_upf_line_width` variable is set to 30 and the `mv_output_upf_line_indent` variable is set to 3.

Example 11-7 UPF File Written With Line-Splitting Enabled

```
set mv_output_upf_line_width 30
set mv_output_upf_line_indent 3
set_port_attributes -ports \
    {instA/power_port} \
    -attribute snps_derived
```

Note:

Power Compiler issues error messages if you set any of these variables to a value less than 0.

Writing and Reading Verilog Netlists With Power and Ground Information

The Power Compiler tool supports the writing and reading of Verilog netlists containing the complete power and ground (PG) supply connection information, including supply connections to the leaf-level library cells.

To write out a Verilog netlist with the complete PG supply connection information, use the `-pg` option of the `write_file` command. For example,

```
dc_shell> write_file -format verilog -pg output top_with_pg.v
```

When you read in a Verilog netlist with the complete PG supply connection information using the `read_verilog -netlist` command, the tool automatically recognizes and restores the PG supply connection information stored in the netlist. If there are any conflicts between the Verilog netlist and the `connect_supply_net` commands in the UPF file, the tool reports the differences as errors.

Power and Ground Supply Connection Syntax

When you specify the power and ground connections in a Verilog module, the Verilog netlist, PG supply connections use the same syntax as the logic signal connections. For example,

```
module test ( A, Z, VDD, VSS );
    input A, Z;
    input VDD;
    input VSS;
    ...
```

When power and ground nets need to cross over levels of the Verilog hierarchy that do not have corresponding crossovers in the UPF hierarchy, the tool punches ports to carry the PG nets into the lower levels of the hierarchy, as in the following example.

```
// VDDINT is not used in any leaf cells in 'top'
module top ( A, Z, VDD, VSS, VDDINT );
    input A;
    input VDD;
    input VSS;
    input VDDINT;
    output Z;
    ...
    mid M1 ( .A( A ), .Z( Z ), .VDDINT( VDDINT ), .VSS( VSS ) );
module mid ( A, Z, VDDINT, VSS );
    ...
```

The tool does not punch ports at the top level. If a supply net drives a cell at the top level, but the net is not explicitly connected to a supply port or a supply set function defined in the UPF file, the supply inputs to those cells are undriven wires.

Each module net has the same name as the UPF supply net corresponding to the PG supply net. Because the names of the supply net does not need to match the name of the supply port, the tool uses the following syntax for the supply net:

```
// corresponds to UPF statement:
// connect_supply_net VDDINT -ports VDD
module test ( A, Z, .VDD( VDDINT ), VSS );
```

Each instance of a cell with PG pins shows the PG connections for the instance, including the pin name and the name of the supply net connected to the pin. The supply pins appear last, after the signal pins, as shown in the following example.

```
...
SIMPLE_PG_CELL U1 (.a(A), .z(Z), .VDD( VDD0 ), .VSS( VSS ));

COMPLEX_PG_CELL U1 (.a(A), .z(Z), .VDD( VDD0 ),
                    .VDDBACKUP( SNPS_ss_SS_P1$primary$power ),
                    .VSS( VSS ));
...
```

Supply Sets

If the UPF file uses supply sets or supply set handles, and these supplies are not resolved to supply nets, the tool writes out the supply set definitions in the Verilog netlist. The supply set definitions contain the SNPS_ss_ prefix and use the dollar character \$ in place of each period character that delimits the fields within a supply set identifier. For example,

```
module test ( A, Z, .VDD1( SNPS_ss_ss1$power ),
              .VDD2( SNPS_ss_pd1$primary$power ),
              .VSS( SNPS_ss_pd1$primary$ground ));
  input A, SNPS_ss_ss1$power,
         SNPS_ss_pd1$primary$power,
         SNPS_ss_pd1$primary$ground;
  output Z;
  wire VDD$; // switched supply
  ...
```

Power Switches

The Power Compiler tool does not instantiate power switch cells. However, power switches have output supply nets that can power other cells. The tool writes out these supply nets without drivers. For example,

```
module top(a, z, VDDA, VDDDB, VSS);
  input a;
  input VDDA;
  input VDDDB;
  input VSS;
  output z;

  mid M1(.a(a), .z(z), .VDDA(VDDA), .VDDDB(VDDDB), .VSS(VSS));
  ...

Module mid(a, z, VDDA, VDDDB, VSS);
  input a;
  input VDDA;
  input VDDDB;
  input VSS;
  wire VDD_SW;
  wire a1;

  SIMPLE_PG_CELL U1 (.a(a), .z(a1), .VDD( VDD_SW ), .VSS( VSS ));
  // The ports in the bot design are punched as follows:
  bot B1 (.a(a1), .z(z), .VDDA(VDDA), .VDDDB(VDDDB), .VSS(VSS), \
         .VDD_SW(VDD_SW));
  ...
```

The Power Compiler tool does not write out the PG nets and ports that do not drive any PG pins of leaf-level cells. The power switch behavior is derived from the UPF description provided with the PG netlist.

Reading Verilog Netlists With Power and Ground Supply Connections

The Power Compiler tool requires PG connections to be represented in UPF format before they can process the design. When the tool reads in a Verilog netlist containing power and ground information, it matches the PG supply connections in the netlist to the UPF supplies and convert these connections into UPF commands. If the tool finds any conflicts, error messages are reported.

The Verilog netlist with PG supply connections must satisfy the following requirements:

- The netlist must contain a complete set of PG connections for all cells in the design, including standard cells.
- The netlist must be created by a Galaxy tool with a valid UPF infrastructure.
- The PG connections must be intact as written by the tool, and not modified by the user.

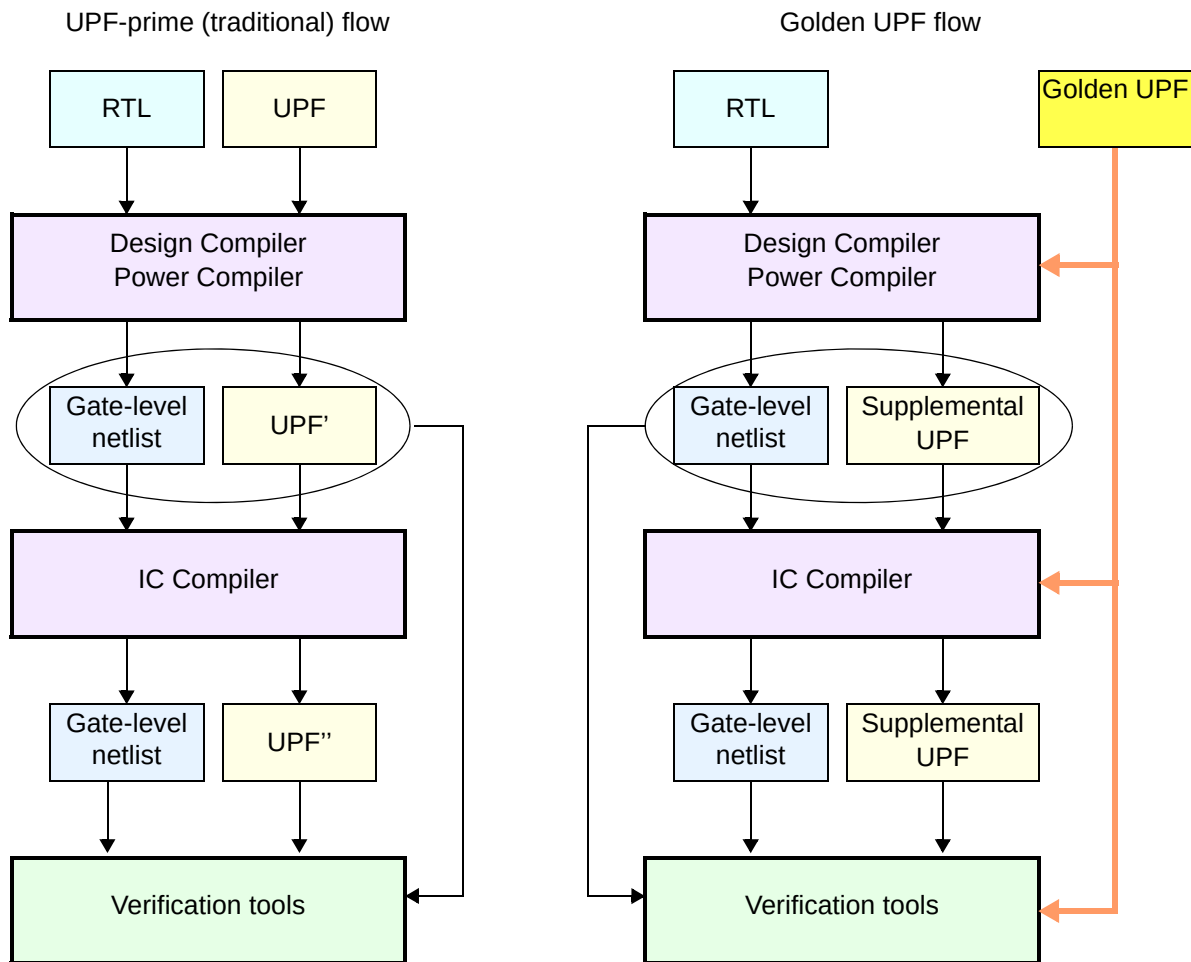
Meeting these requirements ensures that every PG connection in the input netlist corresponds to a UPF supply net as specified in the UPF files, and no new supply nets, supply ports, or supply connections are needed.

The Formality and Verdi NLP tools use the PG Verilog netlist alone to represent the PG connections of the design. These tools do not need to match the PG netlist to UPF supplies, although they might extract other power infrastructure information from the UPF files.

Golden UPF Flow

The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Design Compiler and IC Compiler tools. [Figure 11-43](#) compares the traditional UPF flow with the golden UPF flow.

Figure 11-43 UPF-Prime (Traditional) and Golden UPF Flows



The golden UPF flow maintains and uses the original “golden” UPF file unchanged throughout the flow. The Design Compiler and IC Compiler tools write power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF' or UPF'' file.

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.

- You can optionally use the Verilog netlist to store all the PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

To use the golden UPF flow, you must enable it by setting the following variable:

```
dc_shell> set_app_var enable_golden_upf true
```

After you enable this mode, to execute any UPF commands other than query commands, you must put the commands into a script and execute them using the `load_upf` command. You cannot run the commands individually on the command line or by using the `source` command.

For more information about using the golden UPF mode, see [SolvNet article 1412864](#), “Golden UPF Flow Application Note.”

Reporting Commands for the UPF Flow

The following reporting commands are supported in Power Compiler. These are not UPF standard specified commands.

- `report_dont_touch`
- `report_power_domain`
- `report_level_shifter`
- `report_power_switch`
- `report_pst`
- `report_isolation_cell`
- `report_retention_cell`
- `report_supply_net`
- `report_supply_port`
- `report_target_library_subset`
- `report_mv_library_cells`

For more details, see the command man pages.

UPF-Based Hierarchical Multivoltage Flow Methodology

Design Compiler topographical mode supports flat, top-down, and bottom-up hierarchical UPF design flows. These flows are also supported by Synopsys tools such as IC Compiler, PrimeTime, and Formality. This section describes the UPF portion of the hierarchical design methodology. For basic information about the hierarchical design methodology, see the *Design Compiler User Guide*.

When you synthesize your design using the UPF-based hierarchical flow, specify the voltage for each supply net. Also specify the timing constraints as recommended in the [SolvNet article 026172, "IEEE \(1801\) UPF Based Design Compiler Topographical Technology and IC Compiler Hierarchical Design Methodology."](#)

In the hierarchical implementation of a design, you first determine the physical partition. Follow these guidelines while partitioning your design:

- The scopes of all power domains within a partition must be contained inside the partition.
- For all supply connections inside a partition, supply nets must be specified within the partition.
- The partitions should not be nested.

Steps in the Hierarchical UPF Design Methodology

To implement your design using the Design Compiler hierarchical UPF design methodology, follow these two steps:

1. [Block-Level Implementation](#)
2. [Top-Level Implementation](#)

Each of these steps is described in detail in the following sections:

Block-Level Implementation

Creating the Blocks

Create the block-level and top-level UPF files for the design. To create the blocks, you can use either the top-down approach or the bottom-up approach. The bottom-up approach is preferable because this determines the smallest block that can be compiled independently.

When the individual blocks and the top are synthesized, you can assemble the design either in Design Compiler or in IC Compiler. To assemble the design using IC Compiler, the tool requires the complete design database for the design planning stage. For more details, see [Assembling Your Design](#).

Generating the Block-Level UPF Constraints

To use the hierarchical UPF methodology, your constraint specification in the UPF file must also be hierarchical. You can choose one of the following two ways to create the block-level and top-level UPF files.

- Write the power intent manually in the UPF file for all the blocks, including the top. If required, write the boundary constraints for the blocks.
- Use the `characterize` command to create the block-level UPF constraints as well as the boundary constraints from the full chip UPF description. It is important to remember the following points when you use the `characterize` command to generate the block-level UPF constraints:
 - If your design does not have the control signals at the block-level interfaces and you cannot modify your block level interfaces, you must use the `characterize` command to generate the block-level UPF constraints.
 - By default, the `characterize` command propagates the UPF constraints in the top design to the subblock.

However, if you use this approach, you can perform equivalence checking only on the entire design and not on each hierarchical block.

Note:

All necessary power management control signals should be created manually. They also have to be manually brought into the appropriate block-level interfaces. This is the recommended approach.

Using Manually Created Block-Level UPF Files

When you create the blocks manually, each block and its power intent in the UPF file must be written such that each block can be simulated and synthesized independently. You might have to write the boundary constraints for the blocks to capture any port that does not operate at the same voltage as the rest of the block. If a block contains a power domain, the UPF constraints refer to objects and power supplies only within the block.

Using ETMs and Macros for Block-Level UPF Files

An ETM (Extracted Timing Model) is the Liberty model representation of a design. An ETM captures the UPF information using relevant Liberty attributes. A macro design might be represented using the Liberty model or it might be an IP provided by a vendor. The tool treats macros and ETMs in the same way and does not distinguish between the two types of implementation.

The tool supports ETMs and macros in the UPF hierarchical flow as follows:

- Only the UPF of the interface logic is required for the hierarchical UPF implementation
- You can also provide the full block UPF of the macro or ETM for the tool to automatically extract the interface power intent
- The UPF intent of the macro or ETM design's top power domain is used for design integration

The UPF constructs can be created and referenced at the top-most scope of a macro or ETM. Set the `upf_suppress_etm_model_checking` variable to `true` (default is `false`) if you want to bypass ETM and macro checking when loading a UPF file. For example, if `ublock` is modeled as an ETM, you would do the following:

```
set_app_var upf_suppress_etm_model_checking true
load_upf block.upf -scope ublock
```

Note that UPF constructs defined at the scope of nested logic within a macro or ETM are read and ignored.

Using Design Compiler Generated Block-Level UPF Files

If you use the top-down approach to write your design or if your UPF file is nonmodular, Design Compiler can generate the block-level UPF using the `characterize` command. For the tool to correctly generate the block-level UPF file, your power domain definition and partitioning should comply with the guidelines mentioned in [UPF-Based Hierarchical Multivoltage Flow Methodology](#). The UPF objects in the block should not refer to any object that is above the block in the hierarchy. You should follow these steps to synthesize your design using the hierarchical UPF design methodology:

1. Read the design and the UPF constraints for the entire design.
2. Specify the operating voltages for the supply nets and specify the timing constraints.
3. For each subblock in the design, perform the following tasks:
 - a. Run the `characterize` command.

This command pushes the appropriate timing and power constraints from the top-level to the specified block. The block-level power constraints and the boundary constraints that are specified by the `set_related_supply_net` command are set on the specified block. For more details, see [Characterization of Supply Sets and Supply Nets](#).

The `characterize` command can also automatically set the related supply net on the ports of the block-partition. To avoid voltage violations at the boundary, that can be caused by the automatic setting of related supply net, you must define level-shifter strategies at the block-partition boundary. If you do not want certain ports to be level

shifted, use the `set_level_shifter -no_shift` command. For more details see [Automatic Inference of Related Supply Net](#).

While setting the related supply net, additional checks are performed for voltage violations, availability of the supply net, and so on, and appropriate error and warning messages are issued.

- b. Save the characterized block and the design data.

Set the characterized block as the current instance and use the `write` command to save the characterized block. The command sequence is shown in the following example.

```
characterize BlockA
set current_instance BlockA
write -format ddc -hierarchy -output BlockA.characterized.ddc
```

- c. Remove the block from the top level using the `remove_design -hierarchy` command. When you remove the block, the UPF constraints associated with the block are also removed.
4. When all the subblocks have been characterized, saved in .ddc format, and removed, save the top-level design in .ddc format.

Synthesizing the Blocks

To synthesize each subblock of the hierarchical design, you can read the design in one of the following two methods:

- The RTL file and the manually written UPF file for each block.
- GTECH netlist in the .ddc file for each block, written after the characterization step.

The difference between the two is the clarity of the block-level UPF and the automatic inclusion of boundary constraints when you use the .ddc file generated after the characterization step and the ability to perform hierarchical verification using Formality. The power intent created by the `characterize` command is the same as the manually created UPF file. If you use the RTL design and the manually written UPF file, you should create appropriate boundary constraints.

You then use either the top-down or bottom-up synthesis flow options supported in Design Compiler topographical mode to perform block-level synthesis. For more information, see [SolvNet article 021034, "Hierarchical Flow Support in Design Compiler Topographical Mode."](#)

Top-Level Implementation

Follow these steps to perform the top-level synthesis:

1. Read the block-level designs.

The block-level design can be any one of the following types:

- A synthesized block-level design
- A block abstraction created either in Design Compiler or in IC Compiler

Specify the blocks using the `set_top_implementation_options` command.

2. Read the top-level design in either of the following formats:

- RTL design and UPF files; use the `load_upf` command to read the UPF file
- GTECH netlist in .ddc file format, obtained after removing all the characterized subblocks

Note: When reading the top-level UPF file before the block-level UPF file, you must set the `upf_allow_refer_before_define` variable to `true` to allow loading a top-only UPF file with references to unlinked subblocks within the design. By default, this variable is set to `false`.

3. Run the `propagate_constraints -power_supply_data` command.

This command propagates all the block-level constraints to the top-level, including the block abstractions created in Design Compiler or IC Compiler, that contain UPF data.

4. Synthesize the top-level design.

For the block abstractions created in Design Compiler topographical mode, the tool performs size-only optimization on the block interface logic, including the power management cells.

5. Save the synthesized design and the UPF constraints.

When you save the design in .ddc file format, the UPF constraints are also saved in the file. To save the UPF constraints separately, use the `save_upf` command. To save the complete UPF information, use the `save_upf -full_upf` command. To save only the top-only UPF, use the `save_upf` command. You can also set the `upf_block_partition` variable to specify the name of a block or a list of blocks whose UPF information you do not want to save. For instance, if you have a subblock named `block1` and you want to save the top UPF without `block1`, do the following:

```
set_app_var upf_block_partition block1
save_upf
```

Note that if `block1` is an ETM, macro, or any other black box, you do not need to specify it in the `upf_block_partition` list since the `save_upf` command automatically skips these blocks.

Completing these steps completes the synthesis of your design using the Design Compiler hierarchical UPF flow. Using the synthesized design, you can continue the flow in IC Compiler. For more details on assembling your design for the subsequent steps in IC Compiler, see [Assembling Your Design](#).

Assembling Your Design

To continue with the hierarchical flow in IC Compiler, you can assemble your design either in Design Compiler or in IC Compiler. Note that you must explicitly ensure that the block-level UPF constraints are available in the top-level design during the optimization step of the top-level. You do this using the `propagate_constraints -power_supply_data` command. Use the following steps to assemble your design in Design Compiler for use in the further flow in IC Compiler:

1. Read all the synthesized subblocks.
2. Set the top-level design as the current design.
3. Link the design using the `link` command.
4. Use the `propagate_constraints -power_supply_data` command for all the block-level UPF constraints to be available at the top-level.
5. Save the design. This saved design is the full chip design database that you can use to start the design planning step in IC Compiler.

For more information, see [SolvNet article 026172, "IEEE 1801 \(UPF\) Based Design Compiler Topographical Technology and IC Compiler Hierarchical Design Methodology."](#)

Characterization of Supply Sets and Supply Nets

The following sections describe criteria for characterization of the supply sets and domain-independent supply nets and how they are characterized during the hierarchical UPF flow.

Criteria for Characterization

A supply set or a domain-independent supply net of a block is characterized when it is,

- The primary, default retention, default isolation supply of the power domain of the block
- The supplies specified in the retention or isolation strategies of the power domain of the block
- A supply that is specified for the power switch of the power domain of the block
- An exception supply that is connected to the cells in the power domain of the block

- An extra supply of the power domain, defined by using the `extra_supplies_#` keyword
- A supply set that is connected to the supply ports that are defined inside the block
- A supply set that is the related supply for the ports of the block

Note:

In this case, the supplies are characterized even if they are the restricted supplies in the top-level power domain of the block being characterized. This is because, the block can contain an unrestricted feedthrough supply that passes through power domains.

Characterization of Supply Sets

While partitioning a block, the supply sets defined in the block and in lower levels of hierarchy are moved to the block. The supply sets and domain-independent supply nets are handled similarly because supply sets are also inherently domain-independent.

When a `repeater_supply` attribute is specified in the path of an isolation strategy defined using the `-source`, `-sink`, or `-diff_supply_only` option, the value of the `repeater_supply` attribute is used to derive the value of the `iso_source` and `iso_sink` attributes at the boundary of the block.

Updates at the Block Level

During characterization, at the block level,

- Two supply ports and a supply set are created. The supply ports are connected to the power and ground functions of the supply set.
- To distinguish the supply ports created by the `characterize` command, the newly-created supply ports are marked with the `snps_derived` UPF attribute. So, each supply port created by the `characterize` command has an associated `set_port_attributes` command in the block-level UPF file.
- If you have defined power states for the supply sets for the block-level, using the `add_power_state` command, during characterization, the tool writes the `add_port_state` command for the created port.

Updates at the Top Level

At the top level, and in the UPF file for the top level, two ports are created, which are connected to the power and ground functions of the supply set.

Automatic Inference of Related Supply Net

In the top-down hierarchical flow, when you characterize a block, the block-level power constraints as well as the boundary constraints that are specified by the `set_related_supply_net` command are set on the specified block.

The `characterize` command can also automatically set the related supply net on the ports of the block-partition, using the following criteria:

- The direction of the port.
- The location constraint of the isolation and level-shifter strategies.
- Related supply net of the driver or the load cells.
- The `-driver_supply` and `-receiver_supply` options specified with the `set_port_attributes` command.

The `characterize` command can also infer the driver or load to be inserted at the boundaries.

Note:

For the `characterize` command to appropriately infer and set the related supply net, you must explicitly define the level-shifter and isolation strategies before running the `characterize` command, if you have voltage violations.

[Table 11-10](#) shows the related supply net inferred by the Power Compiler tool when you define only the level-shifter strategy, and not the isolation strategy, to overcome the voltage violations at the boundary pins.

Table 11-10 Related Supply Net With Level-Shifter Only Strategy

Port direction	Level-shifter strategy	Related supply net inferred by Power Compiler
Input	self	Outside or driver supply net. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	Inside or load supply net.
Output	self	Outside or load supply net. If supply is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	Inside or driver supply net.
Input or Output	none or auto	Not supported. UPF-206 error message is issued.

Table 11-11 shows the related supply net inferred by the Power Compiler tool when you define both level-shifter and isolation strategies.

Table 11-11 Related Supply Net With Level-Shifter and Isolation Strategies

Port direction	Level-shifter strategy	Isolation strategy	Related supply net inferred by Power Compiler
Input	self	self	Outside or driver supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	self	parent	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	self	Related supply net is not set and UPF-207 error message is issued.
Input	parent	parent	Inside or load supply net.
Input	none or auto	self or parent	Not supported. UPF-206 error message is issued.
Output	self	self	Outside or load supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	self	parent	Related supply net is not set and UPF-207 error message is issued
Output	parent	self	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	parent	Inside or driver supply.
Output	none/auto	self or parent	Not supported. UPF-206 error message is issued.

Table 11-12 shows the related supply net inferred by the Power Compiler tool when there are no voltage violations at the boundary pins.

Table 11-12 Related Supply Net With No Voltage Violations at the Boundary Pins

Port direction	Isolation strategy	Related supply net inferred by Power Compiler
Input	self	Outside or driver supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Input	parent	Isolation power supply. If supply net is not available, use the inside or load supply.
Input	none	Outside or driver supply. If supply net is not available, use the inside or load supply.
Output	self	Isolation power supply. If supply net is not available, related supply net is not set and UPF-208 error message is issued.
Output	parent	Inside or driver supply.
Output	none	Outside or load supply. If supply net is not available, use the inside or load supply.

Note:

If voltage violations are across two blocks that have to be characterized, define the level-shifter strategies for both the blocks. To avoid level-shifter redundancy, use the `-no_shift` option of the `set_level_shifter` command. If the violations are across multiple blocks, specify the list of pins while defining the level shifter strategy with the `-no_shift` option.

Top-Level Design Integration

After the blocks are characterized, these blocks can be integrated into the top-level designs, multiple times. Use the `propagate_constraints` command to integrate the characterized blocks to the top level.

Power Domain Merging

While merging the power domain to the top level, the `propagate_constraints` command ensures that equivalent supply sets, nets, and ports are present at the top level. In addition, their connectivity should be equivalent at the top level. The tool issues the UPF-168 error message when equivalence is not found.

During integration, the block-level ports that have the `snps_derived` UPF attributes are substituted by their equivalent top-level ports and supply nets or supply sets.

Switch Cell Matching

When a power switch cell exists in the blocks, a matching switch cell must exist at the top level for the domains to be merged. It is an error to match a switch cell from the block to a switch cell in the top level, that is already matched.

When Power Compiler merges domains that contain switch cells, the following rules apply:

- A switch cell in the top level should have a unique switch cell in the domain being merged. The switch cells being merged should also have equivalent
 - Input supply nets
 - Control and acknowledge signals separated by buffer or inverter pairs
 - Voltage setting on the output supply nets
 - Port states, including the state names, state value, primary domain, and so on

Also, the output supply nets must have similar connectivity with the other supply nets in the design.

- When the domain has multiple equivalent switch cells, the first matching switch cell is used.

Legacy Blocks

A conflict can arise when a *legacy block* is used in a design with *domain-independent* supply nets. To prevent such a conflict, you can set the legacy block's `legacy_block` design attribute to `true`. This converts all power domains of the legacy block to be *fully restricted*, so that the legacy block can no longer use any domain-independent supply nets declared in the scopes above the block.

A *domain-independent* supply net is a supply net that is available to any power domain defined at or below the scope of the supply net, as long such domains are not *restricted*. In other words, the supply net was created by a `create_supply_net` command without the `-domain` option. For example,

```
dc_shell> create_supply_net SN1
```

Conversely, a domain-dependent supply net is a supply net that is available only to the domain for which it is defined. In other words, the supply net was created by a `create_supply_net` command with the `-domain` option. For example,

```
dc_shell> create_supply_net SN2 -domain PD2
```

The supply net is created in domain PD2 and cannot be used in other domains.

A *restricted* power domain is a power domain that is restricted to use only certain supply sets. This restriction results from usage of the `extra_supplies` keyword with the `-supply` option of the `create_power_domain` command. For example,

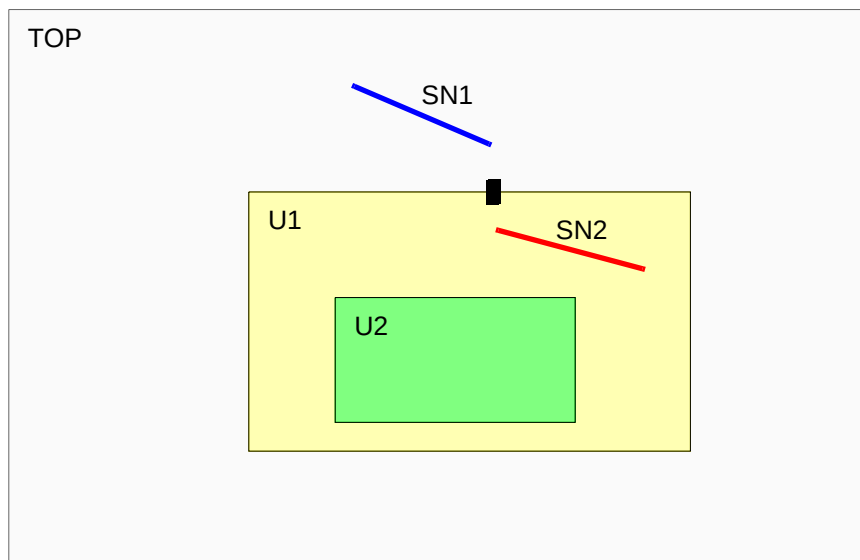
```
dc_shell> create_power_domain PD3 -elements U3 \
          -supply {extra_supplies_0 SS1}
```

The power domain PD3 is restricted to using only the SS1 supply set.

A *legacy block* is a block designed before the introduction of supply sets, using only domain-dependent supply nets, when there was no consideration of possible usage of domain-independent supply nets from a higher level of the design hierarchy. A legacy block does not use or define any domain-independent supply nets or supply sets.

When a legacy block is used in a newer design containing supply sets, a conflict can arise with a situation like the one shown in [Figure 11-44](#).

Figure 11-44 Legacy Block Used in a Top-Level Design



In this diagram, U1 is a legacy block with domain-dependent supply net SN2. An instance of this block is used in the top-level design, TOP, which has domain-independent supply net SN1. U1 contains a lower-level block, U2. Because supply net SN1 is domain-independent, it is available for use in the gray, yellow, and green domains. On the other hand, because supply net SN2 is domain-dependent, it is available for use only in the yellow domain.

If the two supply nets are connected together through a supply port on U1, the availability of the combined net in U2 is undefined. It could lead to the usage of the supply net in U2, which would be incorrect.

To clearly specify that this type of connection is not allowed, you can declare U2 to be a legacy block by using the following command:

```
dc_shell> set_design_attributes -elements U1 \
          -attribute legacy_block true
```

This converts all power domains of block U1 to fully restricted domains so that those domains can no longer use the domain-independent supply nets declared in scopes above the block. The tool achieves this effect by using the `-supply` option of the `create_power_domain` command when the block is read in.

For example, the tool changes this command:

```
create_power_domain PD -elements U2
```

to this command:

```
create_power_domain PD -elements U2 -supply {extra_supplies ""}
```

Because there are no supply sets listed between the quotation marks in the `-supply` option, the domain becomes fully restricted and does not allow the usage of any supply sets defined at higher levels of the design, thereby preventing any supply set availability conflict from arising.

No domain-independent supply nets or supply sets can be defined or used inside a legacy block or any of its lower-level blocks, and no supply set handles can be used. Any lower-level blocks below a legacy block must also be legacy blocks.

The `propagate_constraints` command supports the usage of legacy blocks. The following script shows an example of the flow.

```
read_verilog top_only.verilog
load_upf top_only.upf
read_verilog my_legacy_block.verilog
load_upf my_legacy_block.upf
current_design top
#Mark the block as legacy block
set_design_attributes -elements {U2} -attribute legacy_block TRUE
#propagate_constraints makes all the domains in the block restricted
propagate_constraints -design my_legacy_block
```

Note:

The `characterize` command is not supported for legacy blocks.

12

Library Setup for Power Optimization

This chapter describes library setup required for performing power optimization:

- [Basic Library Requirements for Multivoltage Designs](#)
- [Library Usage in Multicorner-Multimode Designs](#)
- [Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells](#)

Basic Library Requirements for Multivoltage Designs

To synthesize your multivoltage design using Power Compiler, the target libraries you use must conform to the Liberty open library rules. The target libraries should also support special cells such as clock-gating cells, level-shifters, isolation cells, retention registers, and always-on buffers and inverters. To support synthesis of multivoltage designs, the tool also supports multiple libraries characterized at different voltages.

Power and Ground Pin Syntax

If the target library that you specify complies with the power and ground (PG) pin Liberty library syntax, Power Compiler uses this information during the synthesis process. However, if your target library does not contain PG pin information, you can convert it into PG pin library format. For more information, see [Converting Libraries to PG Pin Library Format](#).

Converting Libraries to PG Pin Library Format

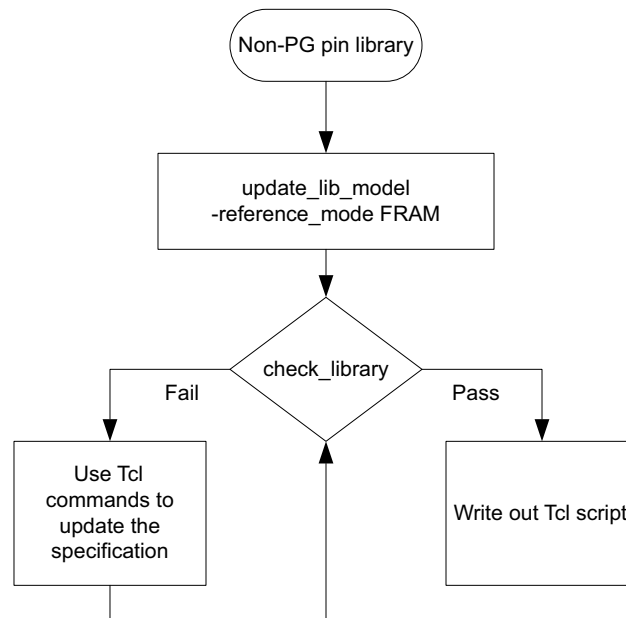
If the libraries that you specify do not contain PG pin information, you can define them in the library to conform to PG pin Liberty syntax. These are discussed in detail in the following sections:

- [Using FRAM View](#)
- [Using Tcl Commands](#)
- [Tcl Commands for Low-Power Library Specification](#)

For more information, you can also see [SolvNet article 029641](#), “On-the-Fly Low-Power Library Specification.”

Using FRAM View

In the Design Compiler topographical mode, you can use the FRAM view as the reference for converting your library to the PG pin library format. You must set the `mw_reference_library` variable to the location of the Milkyway reference libraries. Use the `update_lib_model` command to convert your library to the PG pin library format. The tool uses the PG pin definitions available in the FRAM view of the Milkyway library for the conversion. This is the default behavior. [Figure 12-1](#) shows the steps involved in converting non-PG pin library to a PG pin library.

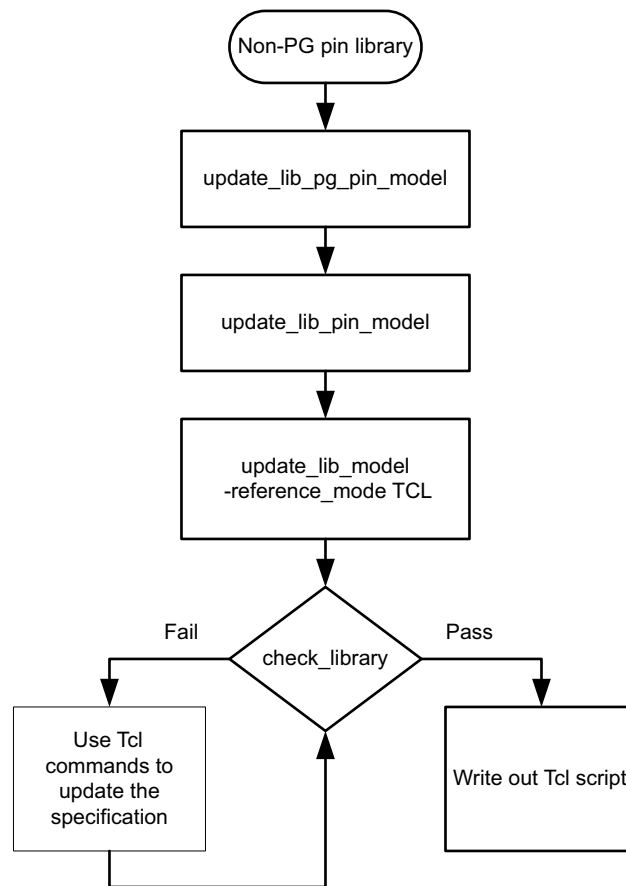
Figure 12-1 Conversion of a non-PG Pin Library to a PG Pin Library Using FRAM View

To ensure that the PG pin library that is created is complete, use the `check_library` and `report_mv_library_cells` commands. If the PG pin library is not complete, run the library specification Tcl commands to complete the library creation. For more information, see [Tcl Commands for Low-Power Library Specification](#).

Using Tcl Commands

When your library files are not in the PG pin library syntax and you do not have the FRAM view of Milkyway library, you can use the following Tcl commands to specify the necessary information required for deriving the PG pin details, as shown in [Conversion of Non-PG Pin Library to PG Pin Library Using Tcl Commands](#).

- `update_lib_voltage_model`
This command sets the voltage map for the specified library.
- `update_lib_pg_pin_model`
This command sets the PG pin map for the specified library cell.
- `update_lib_pin_model`
This command sets the pin map for the specified library cell.

Figure 12-2 Conversion of Non-PG Pin Library to PG Pin Library Using Tcl Commands

These Tcl commands specify the library requirements that are used while converting the libraries to PG pin format.

Run the `update_lib_model -reference_mode TCL` command to convert your libraries to PG pin library format. To check if your newly created PG pin library is complete, run the `check_library` command. If your newly created PG pin library contains conflicts or is incomplete, you can run the library specification Tcl commands to complete the library specification. For more details, see [Tcl Commands for Low-Power Library Specification](#).

Tcl Commands for Low-Power Library Specification

When you convert your library to PG pin format, if the newly created library file is complete, you can start using the library for the low-power implementation of your design. However, if your library contains power management cells and the modeling is not complete, you can use the following Tcl commands to complete your library specifications. These commands specify the library voltage and PG pin characteristics.

- `set_voltage_model`

This command sets the voltage model on the specified library by updating the voltage map in the library.

- `set_pg_pin_model`

This command defines the PG pins for the specified cell.

- `set_pin_model`

This command defines the related power, ground, or bias pins of the specified pin of the library.

For more details, see the command man page and the Library Checking Chapter in the *Library Quality Assurance System User Guide*.

Macro Cells with Fine-Grained Switches

Power Compiler supports macro cells with fine-grained switches, which have the following attribute settings in the PG pin definition in the library:

- The `direction` attribute is `internal`.
- The `pg_type` attribute is either `internal_power` or `internal_ground`.
- The `pg_function` attribute is defined.
- The `switch_function` attribute is defined.
- The `switch_cell_type` attribute of the macro is `fine_grain`.
- The `switch_pin` attribute is set to `true` for the control port.

Use the `connect_supply_net` command to connect to the internal PG pins of these macro cells. However, supply nets connected only to the internal PG pins of these macro cells cannot be used for level-shifter insertion and always-on synthesis, unless the following conditions are true:

- The supply net is the primary supply of the power domain.
- The supply net is specified by the isolation strategy of the power domain.

- The supply net is specified by the retention strategy of the power domain.
- The supply net is defined or reused as a domain-dependent supply net of the power domain.
- The supply net is defined with the `extra_supplies_#` keyword.

You can use the `set_voltage` command to set the operating voltage on the internal PG pins of the macro cells with fine-grained switches. If you do not set the voltage on the internal PG pin of the macro cell, the value of the `voltage_name` attribute of the PG pin is used as the operating voltage.

For more details, see the `set_voltage` command man page.

Library Usage in Multicorner-Multimode Designs

The following sections discuss how to handle libraries properly in multicorner-multimode designs:

- [Link Libraries With Equal Nominal PVT Values](#)
- [Distinct PVT Requirements](#)
- [Automatic Detection of Driving Cell Library](#)
- [Relating the Minimum Library to the Maximum Library](#)
- [Unique Identification of Libraries Based on File Names](#)

Link Libraries With Equal Nominal PVT Values

The link library lists all of the libraries that are to be used for linking the design for all scenarios. Furthermore, because several libraries are often intended for use with a particular scenario, such as a standard cell library and a macro library, Design Compiler automatically groups the libraries in the link library list into sets and identifies which set must be linked with each scenario.

Library grouping is based on their PVT values. Libraries with the same PVT values are grouped into the same set. The tool uses the PVT value of a scenario's maximum operating condition to select the appropriate set for the scenario.

If the tool finds no suitable cell in any of the specified libraries, an error is reported as shown in the following example,

```
Error: cell TEST_BUF2En_BUF1/Z (inx4) is not characterized
for 0.950000V, process 1.000000, temperature -40.000000. (LIBSETUP-001)
```

You should verify the operating conditions and library setup. If you do not correct this error, optimization is not performed.

Link Library Example

Table 12-1 shows the libraries in the link library list, their nominal PVT values, and the operating condition (if any) specified in each library. The design has instances of combinational, sequential, and macro cells.

Table 12-1 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Combo_cells_slow.db	1/0.85/130	WORST (1/0.85/130)
Sequentials_fast.db	1/1.30/100	None
Macros_fast.db	1/1.30/100	None
Macros_slow.db	1/0.85/130	None
Combo_cells_fast.db	1/1.30/100	BEST (1/1.3/100)
Sequentials_slow.db	1/0.85/130	None

To create a scenario s1 with the cell instances linked to the Combo_cells_slow, Macros_slow, and Sequential_slow libraries, you run:

```
dc_shell-topo> create_scenario s1
dc_shell-topo> set_operating_conditions -max WORST -library slow.db:slow
```

Note:

Specifying the `-library` option with the `set_operating_conditions` command helps the tool identify the correct PVT for the operating conditions. The PVT of the maximum operating condition is used to find the correct matches in the link library list during linking.

Using this linking scheme, you can link libraries that do not have operating condition definitions. The scheme also provides the flexibility of having multiple library files (for example, one for standard cells, another for macros).

Inconsistent Libraries Warning

When you use multiple libraries, if the library cells with the same name are not functionally identical or do not have identical sets of library pins (same name and order), a warning is issued, stating that the libraries are inconsistent.

You should run the `check_library` command before running a multicorner-multimode flow, as shown in the following example:

```
set_check_library_options -mcmm  
check_library -logic_library_name {a.db b.db}
```

When you use the `-mcmm` option with the `set_check_library_options` command, the `check_library` command performs multicorner-multimode specific checks such as the operating condition or power-down consistencies. When inconsistencies are detected, the tool generates a report. In addition, the tool also issues the following summary information message:

```
Information: Logic library consistency check FAILED for MCM.  
(LIBCHK-360)
```

To overcome the LIBCHK-360 messages, you must check the libraries and the report to identify the cause for the inconsistency. The man page of the LIBCHK-360 information message describes possible causes for the various library inconsistencies.

Setting the dont_use Attribute on Library Cells

When you set the `dont_use` attribute on a library cell, the multicorner-multimode feature requires that all characterizations of this cell have the `dont_use` attribute. Otherwise, the tool might consider the libraries as inconsistent. You can use the wildcard character to set the `dont_use` attribute as follows:

```
set_dont_use */AN2
```

When library cells with a `dont_use` attribute, have a pin order that does not match exactly in the libraries of various corners, Power Compiler continues with the flow, without issuing any error or warning messages. If you remove the `dont_use` attribute of these cells, the tool issues the MV-087 error messages.

Note:

You do not have to issue the command multiple times to set the `dont_use` attribute on all characterizations of a library cell.

Distinct PVT Requirements

If the maximum libraries associated with each corner (scenario) do not have distinct PVT values, the cell instances are linked incorrectly, which results in incorrect timing values. This happens because the nominal PVT values that are used to group the link libraries into sets, group the maximum libraries of different corners into one set. Consequently, the cell instances are linked to the first cell with a matching type in that set (for example, the first

AND2_4), even though the `-library` option is specified for each of the scenario-specific `set_operating_conditions` commands. That is, the `-library` option locates the operating condition and its PVT values but not the library to link.

The following two tables and the following script demonstrate the problem:

[Table 12-2](#) shows the libraries in the link library, listed *in order*; their nominal PVT values; and the operating condition that is specified in each library.

Table 12-2 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Ftyp.db	1/1.30/100	WORST (1/1.30/100)
Typ.db	1/0.85/100	WORST (1/0.85/100)
TypHV.db	1/1.30/100	WORST (1/1.30/100)
Holdtyp.db	1/0.85/100	BEST (1/0.85/100)

[Table 12-3](#) and the script commands that follow show the operating condition specification for each of the scenarios.

Table 12-3 Scenarios and Their Operating Conditions

	Scenarios			
	s1	s2	s3	s4
Maximum Operating Condition (Library)	WORST (Typ.db)	WORST (TypHV.db)	WORST (Ftyp.db)	WORST (Typ.db)
Minimum Operating Condition (Library)	None	None	None	BEST (HoldTyp.db)

```
create_scenario s1
set_operating_conditions WORST -library Typ.db:Typ
create_scenario s2
set_operating_conditions WORST -library TypHV.db:TypHV
create_scenario s3
set_operating_conditions WORST -library Ftyp.db:Ftyp
create_scenario s4
set_operating_conditions \
  -max WORST -max_library Typ.db:Typ \
  -min BEST -min_library HoldTyp.db:HoldTyp
```

The tool groups the Ftyp.db, and TypHV.db libraries into a set with Ftyp.db as the first library in the set. Therefore, the cell instances in scenario s2 are not linked to the library cells in TypHV.db, as intended. Instead, they are linked to the library cells in the Ftyp.db library, assuming that all the libraries include the library cells required to link the design.

Ambiguous Libraries Warning

When you use multiple libraries, if any of the libraries with same-name cells have the same nominal PVT, a warning is issued, stating that the libraries are ambiguous. The warning also states which libraries are being used and which are being ignored.

Automatic Detection of Driving Cell Library

In multicorner-multimode flow, the operating condition setting is different for different scenarios. To build the timing arc for the driving cell, different technology libraries are used for different scenarios. You can specify the library using the `-library` option of the `set_driving_cell` command. But specifying the library is optional because the tool can automatically detect the driving cell library.

When you specify the library using the `-library` option of the `set_driving_cell` command, the tool searches for the specified library in the link library set. If the specified library exists, it is used. If the specified library does not exist in the link library, the tool issues the UID-993 error message as follows:

```
Error: Cannot find the specified driving cell in memory.(UID-993)
```

When you do not use the `-library` option of the `set_driving_cell` command, the tool searches all the libraries for the matching operating conditions. The first library in the link library set, that matches the operating condition is used. If no library in the link library set matches the operating condition, the first library in the link library set, that contains the matching library cell is used. If no library in the link library set contains the matching library cell, the tool issues the UID-993 error message.

Relating the Minimum Library to the Maximum Library

The `set_min_library` command is not scenario-specific. This implies that if you use this command to relate a minimum library to a particular maximum library, that relationship applies to all scenarios.

Table 12-4 Unsupported Multiple Minimum Library Configuration

	Scenarios	
	s1	s2
Maximum library	Slow.db	Slow.db
Minimum library	Fast_0yr.db	Fast_10yr.db

For example, you could not relate two different minimum libraries—for example, `Fast_0yr.db` and `Fast_10yr.db` – with the maximum library, `Slow.db`, in two separate scenarios. The first minimum library that you specify would apply to both scenarios. [Table 12-4](#) shows the *unsupported* configuration.

Note, however, that a minimum library can be associated with multiple maximum libraries. As shown in the example in [Table 12-5](#), the minimum library `Fast_0yr.db` is paired with both the maximum library `Slow.db` of scenario 1 and the maximum library `SlowHV.db` of scenario 2.

Table 12-5 Supported Minimum-Maximum Library Configuration

	Scenarios	
	s1	s2
Maximum Library	Slow.db	SlowHV.db
Minimum Library	Fast_0yr.db	Fast_0yr.db

Unique Identification of Libraries Based on File Names

Two libraries with the same name can be uniquely identified if their file names, which precede the library names, which are colon-separated, are unique. For example, the library `ABC.db:stdcell` (where `ABC.db` is the library file name and `stdcell` is the library name) is identifiable with respect to the library `DEF.db:stdcell`.

However, two libraries that have the same file name and library name but reside in different directories are not uniquely distinguishable. The following two libraries are not uniquely distinguishable:

```
/remote/snps/testcase/LIB/fast/ABC.db
```

```
/remote/snps/testcase/LIB/slow/ABC.db
```

Automatic Inference of Operating Conditions for Macro, Pad, and Switch Cells

In multivoltage and multivoltage-multimode designs, as designs increase in size and complexity, manually specifying the operating conditions and linking them with the appropriate library cells with matching operating conditions becomes difficult. For these types of designs, it is useful to automatically infer the operating conditions, especially for special cells such as multirail pad cells, macro cells and switch cells. When the operating condition set on the design does not match the operating condition of the cell rails or when the design operating condition does not have rails, Power Compiler issues a LIBSETUP-001 error message.

You can disable the automatic operating conditions inference by explicitly setting the operating conditions.

Note:

Power Compiler does not perform automatic operating condition inference for standard cells. The operating conditions of the standard cells should match exactly with the operating conditions of the design.

Using the `set_opcond_inference` Command

Use the `set_opcond_inference` command to specify the operating condition.

Use the `-level` option specifies the degree to which the inferred operating condition can deviate from the operating condition of the design. The value that you can specify with this option are EXACT, UNIQUE_RESOLVED, CLOSEST_RESOLVED, or CLOSEST_UNRESOLVED. When you do not specify this option, the default is CLOSEST_RESOLVED. For more information, see [Deviating from the Inferred Operating Condition and Its Impact](#).

You must use one of `-level` and `-match_process_temperature` options. The tool issues a LIBSETUP-751 information message when the operating conditions are successfully inferred on a cell instance.

For multicorner-multimode designs, the `set_opcond_inference` command applies to all corners and scenarios of the design. To report the settings specified for the operating condition inference, use the `report_opcond_inference` command.

Deviating from the Inferred Operating Condition and Its Impact

The level value you specify with the `-level` option of the `set_opcond_inference` command determines how much the inferred operating condition can deviate from the operating condition of the design. When you set a higher deviation, the probability of automatic operating condition inference is higher, resulting in a smaller number of LIBSETUP-001 error messages. This also implies less accurate timing and power results. The following table summarizes the level values that you can specify with the `-level` option and its impact on the automatic operating condition inference:

Level value specified with the <code>-level</code> option	Degree of deviation in the inferred operating condition and its impact
EXACT	Operating condition inferred is exact. This results in no inference at all. Timing is exact.
UNIQUE_RESOLVED	Operating condition is inferred for the library cell whose name matches exactly with the cell reference name in the design. You cannot choose a different library cell. Timing can be incorrect. You do not encounter LIBSETUP-001 error messages.
CLOSEST_RESOLVED	This is the default. If multiple library cells are available, library cell with a matching reference name whose operating condition is closest to the design is chosen. Choosing this operating condition can cause inaccurate timing.
CLOSEST_UNRESOLVED	The library cell whose operating condition is closest to the design is chosen. The library cell name need not match exactly with the cell reference name in the design.

The details of the behavior of the tool when you set a specific level value with the `-level` option of the `set_opcond_inference` command are described in this section:

- **EXACT**
When you set the level value to EXACT, the automatic operating condition inference is not performed.
- **UNIQUE_RESOLVED**
The tool performs a name based search in the target libraries. If multiple library cells match with the cell name, the tool does not perform the inference. However, if the cell is

present in a unique library file and no other library contains the cell, the operating condition is inferred. Otherwise, operating condition is not inferred on the cell and a LIBSETUP-001 error message is issued.

- **CLOSEST_RESOLVED**

This is the default, when you do not specify the `-level` option of the `set_opcond_inference` command.

For each macro cell, pad cell, or switch cell instance, the tool finds the set of library cells with the same name. If multiple library cells with the same name are found, the tool chooses a single library cell based on the matching PVT values. The cells with exception connections, whose supply net voltage does not match the rail voltages in the library, are also also considered for operating conditions inference.

For cells with exception connections, the tool chooses the library cell with maximum number of rail voltages that match the supply net voltage of the instance. If there are multiple library cells with maximum number of rail voltages that match the supply net voltage of the instance, the inference fails and the tool issues a LIBSETUP-001 error message.

The pad cells in the library whose rail voltages do not match the supply voltage on the port because of the settings of the `set_port_attributes` or the `set_related_supply_net` command are eliminated from operating conditions inference. However, when the tool finds that such eliminations can cause potential LIBSETUP-001 errors, it reconsiders the eliminated cells for operating conditions inference.

Within this set of library cells that are considered for inference, the tool groups the library cells in the following order of priority:

1. The PVT values of the library cell match the PVT values of the design.
2. The process, temperature, and voltage values from one of the rail voltages match the PVT values of the design.
3. The temperature and voltage values of the library cell match the temperature and voltage values of the design.
4. The temperature and voltage from one of the rail voltages match the PVT values of the design.
5. The process and voltage values of the library cell match the process and voltage values of the design.
6. The process and voltage from one of the rail voltages match the PVT values of the design.
7. The voltage value of the library cell matches the voltage value of the design.
8. The voltage value from one of the rail voltages match the voltage value of the design.

9. The process and temperature values of the library cell matches the process and temperature values of the design.
10. None of the process, voltage, and temperature values of the library cell match the process, voltage, and temperature values of the design.

After the library cells are grouped, the tool inspects each group in the order mentioned above. The inference is terminated for the following situations:

- None of the groups contain exactly one cell.
- None of the groups contain any library cell.

When Power Compiler finds a group that contains exactly one cell, the tool chooses the library cell and uses the PVT values of that cell as the operating condition of the associated macro, pad, or switch cell.

- **CLOSEST_UNRESOLVED**

The tool groups the library cells based on the matching names, as in **CLOSEST_RESOLVED**. The tool then picks the first library cell from the first non-empty group of library cells. It then sets the operating condition of the library cell on the specific cell instance and links the cell instance to the library cell.

13

Power Optimization in Multicorner-Multimode Designs

This chapter describes the support for multicorner-multimode technology in Design Compiler Graphical, in the following sections:

- [Optimizing Multicorner-Multimode Designs](#)
- [Reporting Commands](#)
- [Script Example for Multicorner-Multimode Flow](#)

Optimizing Multicorner-Multimode Designs

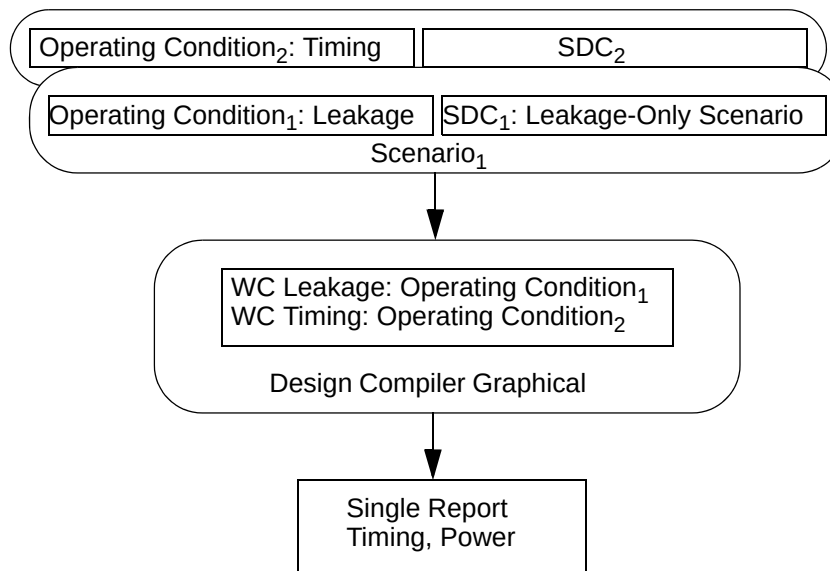
Designs that can be synthesized using multiple operating conditions and in multiple modes are called multicorner-multimode designs. Design Compiler Graphical extends the topographical technology to analyze and optimize these designs. The multicorner-multimode feature also provides ease-of-use and compatibility between flows in Design Compiler and IC Compiler.

For more information about multicorner-multimode concepts and features, see the *Design Compiler User Guide* and *IC Compiler Implementation User Guide*.

Optimizing for Leakage Power

Figure 13-1 shows how to set various constraints on different scenarios of a multicorner-multimode design.

Figure 13-1 Setting Different Constraints on Different Scenarios



Typically, in a multicorner-multimode design, leakage power optimization and timing optimization are done on different corners. Therefore, the worst case leakage corner can be different from the worst case timing corner. To perform leakage power optimization on specific corners, set the leakage power option on specific scenarios of the multicorner-multimode design by using the `set_scenario_options` command as follows:

```
set_scenario_options -scenarios S1 \
  -setup false \
  -hold false \
  -leakage_power true
```


Note:

The `get_dominant_scenarios` command is not supported in Design Compiler Graphical.

When you optimize for leakage power in multiscorner-multimode designs,

- Define the leakage power option on specific scenarios targeted for leakage power optimization.
- Leakage and timing optimizations can be performed concurrently across multiple scenarios.
- The worst case leakage corner is different from the worst case timing corner.
- Do not use the `set_leakage_optimization` command inside a scenario. This command is supported only for non multiscorner-multimode designs.

If no leakage scenario is defined, the average leakage value of all the scenarios is used for leakage optimization.

When you use the `set_multi_vth_constraint` command, you must specify a leakage corner using the `set_scenario_options -scenarios` command.

The following example shows how leakage power is specified on a multiscorner-multimode design. In this example, leakage power optimization is performed only for `scenario_1` and `scenario_3` because the `-leakage_power` option is true:

```
set_scenario_options -scenarios {scenarios_1, scenarios_3} \  
-leakage_power true  
set_scenario_options -scenarios {scenarios_2, scenarios_4} \  
-leakage_power false
```

[Example 13-1](#) shows how to create a scenario and set the leakage power option on the scenario:

Example 13-1 Leakage Power Optimization in a Multicorner-Multimode Design

```
read_verilog top.v
current_design top
link
create_scenario s1
set_operating_conditions WCCOM -library slow.db:slow
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s1.sdc
set_switching_activity -toggle_rate 0.25 \
    -base_clock p_Clk -static_probability 0.015 -type inputs
set_scenario_options -scenarios s1 -setup false -hold false \
    -leakage_power true

create_scenario s2
set_operating_conditions BCCOM -library fast.db:fast
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s2.sdc

create_scenario s3
set_operating_conditions TCCOM -library typ.db:typ
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s3.sdc

create_scenario s4
set_operating_conditions NCCOM -library typ2.db:typ2
set_tlu_plus_files -max_tluplus max.tlu_plus -tech2itf_map tech.map
read_sdc ./s4.sdc
set_scenario_options -scenarios s4 -setup false -hold false \
    -leakage_power true

report_scenarios
compile_ultra -scan -gate_clock
report_power -scenarios [all_scenarios]
report_timing -scenarios [all_scenarios]
report_scenarios
report_qor
report_saif
```

Optimizing for Dynamic Power Using Low-Power Placement

To perform dynamic power optimization in a multicorner-multimode design, use the `set_scenario_options -dynamic_power true -setup true` command. This command performs scenario-specific dynamic power optimization in a multicorner-multimode design. For multiple dynamic power scenarios, the tool uses the average switching activity calculated from data in the SAIF files when performing optimization.

In the Synopsys physical guidance flow, when you enable the low-power placement feature, the tool performs dynamic power optimization for multicorner-multimode designs. To enable this feature, set the `power_low_power_placement` variable to `true` and specify the dynamic power and setup constraints for the scenario. [Example 13-2](#) shows a script to perform dynamic power optimization in multicorner-multimode designs in the Synopsys physical guidance flow.

Example 13-2 Dynamic Power Optimization in a Multicorner-multimode Design

```
set power_low_power_placement true
current_scenario S1
read_saif -input S1.saif
set_scenario_options -dynamic_power true -setup true
compile_ultra -spg
```

For more information about dynamic power optimization see [Dynamic Power Optimization](#).

Reporting Commands

This section describes the commands that you can use for reporting multicorner-multimode designs.

report_scenarios Command

The `report_scenarios` command reports the scenario setup information for multicorner-multimode designs. The scenario specific information includes the logic library used, the operating condition, and TLUPlus files.

The following example shows a report generated by the `report_scenarios` command:

```
*****
Report : scenarios
Design : DESIGN1
scenario(s) : SCN1
Version: ...
Date    : ...
*****

All scenarios (Total=4): SCN1 SCN2 SCN3 SCN4
All Active scenarios (Total=1): SCN1
Current scenario      : SCN1

Scenario #0: SCN1 is active.
Scenario options:
Has timing derate: No
Library(s) Used:
    technology library name (File: library.db)

Operating condition(s) Used:
    Analysis Type      : bc_wc
    Max Operating Condition: library:WCCOM
    Max Process        : 1.00
    Max Voltage        : 1.08
    Max Temperature: 125.00
    Min Operating Condition: library:BCCOM
    Min Process        : 1.00
    Min Voltage        : 1.32
    Min Temperature: 0.00

Tlu Plus Files Used:
    Max TLU+ file: tlu_plus_file.tf
    Tech2ITF mapping file: tf2itf.map
```

Reporting Examples for Multicorner-Multimode Designs

This section contains report examples for some of the report commands used in multicorner-multimode designs.

report_scenarios

The `report_scenarios` command reports the scenario setup information for multicorner-multimode designs. This command reports all the defined scenarios. The scenario-specific information includes the logic library used, the operating condition, and the TLUPlus files.

The following example shows a report generated by the `report_scenarios` command:

```
*****
Report : scenarios
Design : DESIGN1
scenario(s) : SCN1
Version: ...
Date    : ...
*****

All scenarios (Total=4): SCN1 SCN2 SCN3 SCN4
All Active scenarios (Total=1): SCN1
Current scenario      : SCN1

Scenario #0: SCN1 is active.
Scenario options:
Has timing derate: No
Library(s) Used:
  technology library name (File: library.db)

Operating condition(s) Used:
  Analysis Type      : bc_wc
  Max Operating Condition: library:WCCOM
  Max Process       : 1.00
  Max Voltage       : 1.08
  Max Temperature   : 125.00
  Min Operating Condition: library:BCCOM
  Min Process       : 1.00
  Min Voltage       : 1.32
  Min Temperature   : 0.00

Tlu Plus Files Used:
  Max TLU+ file: tlu_plus_file.tf
  Tech2ITF mapping file: tf2itf.map
```

report_power

The `report_power` command supports the `-scenarios` option. Without the `-scenarios` option, only the current scenario is reported. To report power information for all scenarios, use the `report_power -scenarios [all_scenarios]` command.

Note:

In the multicorner-multimode flow, the `report_power` command does not perform clock tree estimation. The command reports only the netlist power in this flow.

The following example shows the report generated by the `report_power -scenarios` command.

```
*****
Report : power
Design : Design_1
Scenario(s): s1
Version: ...
Date   : ...
*****
```

Library(s) Used: slow (File: slow.db)

Global Operating Voltage = 1.08

Power-specific unit information :

Voltage Units = 1V

Capacitance Units = 1.000000pf

Time Units = 1ns

Dynamic Power Units = 1mW (derived from V,C,T units)

Leakage Power Units = Unitless

Warning: Could not find correlated power. (PWR-725)

Power Breakdown

```
-----
```

Cell	Cell Internal Power (mW)	Driven Net Switching Power (mW)	Tot Dynamic Power (mW) (% Cell/Tot)	Cell Leakage Power(nW)
Netlist Power	4.8709	1.2889	6.160e+00 (79%)	1.351e+05
Estimated Clock Tree Power	N/A	N/A	(N/A)	N/A

```
-----
```

Script Example for Multicorner-Multimode Flow

[Example 13-3](#) shows a basic script example for the multicorner-multimode flow.

Example 13-3 Basic Script to Run a Multicorner-Multimode Flow

```
#.....path settings.....
set search_path ". $DESIGN_ROOT $lib_path/dbs \
  $lib_path/mwlibs/macros/LM"
set target_library "stdcell.setup.ftyp.db \
  stdcell.setup.typ.db stdcell.setup.typhv.db"
set link_library [concat * $target_library \
  setup.ftyp.130v.100c.db setup.typhv.130v.100c.db \
  setup.typ.130v.100c.db]
set_min_library stdcell.setup.typ.db -min_version stdcell.hold.typ.db

#.....MW setup.....
#.....load design.....

create_scenario s1
set_operating_conditions WORST -library stdcell.setup.typ.db:stdcell_typ
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s1.sdc
set_scenario_options -scenarios s1-setup false -hold false \
-leakage_power true

create_scenario s2
set_operating_conditions BEST -library stdcell.setup.ftyp.db:stdcell_ftyp
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s2.sdc

create_scenario s3
set_operating_conditions NOM -library stdcell.setup.ftyp.db:stdcell_ftyp
set_tlu_plus_files -max_tluplus design.tlup -tech2itf_map layermap.txt
read_sdc s3.sdc

set_active_scenarios {s1 s2}
report_scenarios
compile_ultra -scan -gate_clock
report_qor
report_constraint
report_timing -scenarios [all_scenarios]
.
.
insert_dft
.
.
compile_ultra -incremental
```

The multicorner-multimode design in [Figure 13-2](#) and the subsequent example scripts in [Example 13-4](#) and [Example 13-5](#) show how you define your power intent in the UPF file and define the scenarios for a multicorner-multimode multivoltage design.

Multicorner-multimode multivoltage designs are useful in applications such as dynamic voltage and frequency scaling (DVFS). In hierarchical designs, the top-level design is

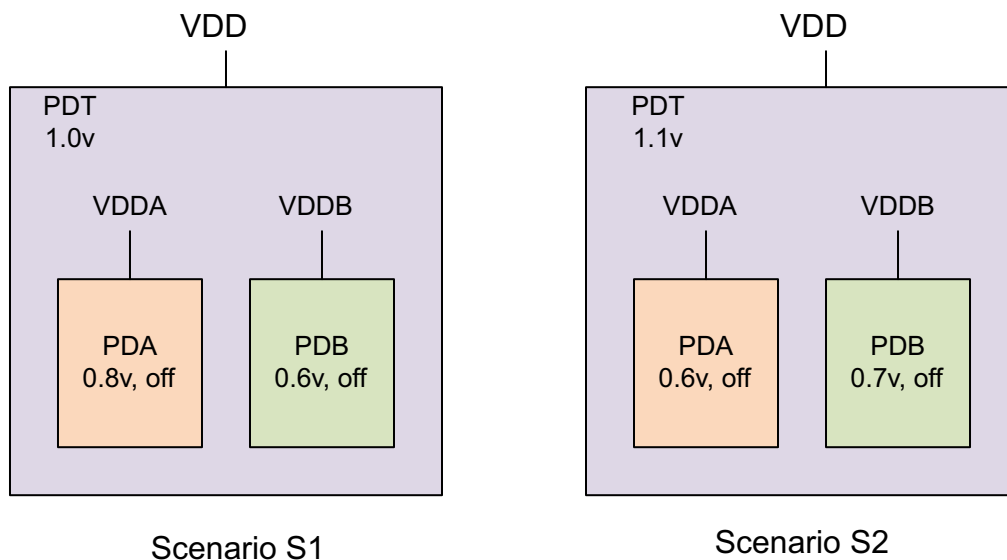
generally optimized at a different voltage and in a different corner than the subdesigns of the hierarchy. The power intent specification can be for the entire design in a single UPF (Unified Power Format) file.

Standard cell and special cell libraries should be available to satisfy all voltages defined across multiple corners.

The design in [Figure 13-2](#) has two scenarios of operation, S1 and S2. In the scenario S1, the power domain PDT operates at 1.0V, while the power domain PDA operates at 0.8V or OFF and power domain PDB operates at 0.6V or OFF. In scenario S2, the power domain PDT operates at 1.1V, while the power domain PDA operates at 0.6V or OFF and power domain PDB operates at 0.7V or OFF.

Although the various subdesigns operate at different voltages, you need only a single UPF file to specify your power intent for the entire design and all its subdesigns. The specific voltages set on the supply nets are scenario-specific and are set by using the `set_voltage` command in each scenario.

Figure 13-2 Multicorner-Multimode Design With Multivoltage



[Example 13-4](#) and [Example 13-5](#) show example scripts using the UPF flow for the multivoltage, multicorner-multimode design in [Figure 13-2](#).

Example 13-4 UPF File Describing Design Intent

```

Example UPF File
## Create Power Domains
create_power_domain PDT -include_scope
create_power_domain PDA -elements PD_PDA
create_power_domain PDB -elements PD_PDB

## Create Supply Nets
create_supply_net VDD -domain PDT
create_supply_net VDDA -domain PDA
create_supply_net VDDB -domain PDB
create_supply_net VSS -domain PDT
create_supply_net VSS -domain PDA -reuse
create_supply_net VSS -domain PDB -reuse

## Create Supply Ports
create_supply_port VDD
create_supply_port VDDA
create_supply_port VDDB
create_supply_port VSS

## Connect supply nets
connect_supply_net VDD -ports VDD
connect_supply_net VDDA -ports VDDA
connect_supply_net VDDB -ports VDDB
connect_supply_net VSS -ports VSS

#### Adding port states
add_port_state VDD -state {HV1 1} -state {HV2 1.1}
add_port_state VDDA -state {LV1 0.8} -state {LV3 0.6} -state {OFF off}
add_port_state VDDB -state {LV2 0.9} -state {LV4 0.7} -state {OFF off}
create_pst top_pst -supplies "VDD VDDA VDDB"
add_pst_state PM1 -pst top_pst -state { HV1 LV1 LV3 }
add_pst_state PM2 -pst top_pst -state { HV1 LV1 OFF }
add_pst_state PM3 -pst top_pst -state { HV1 OFF LV3 }
add_pst_state PM4 -pst top_pst -state { HV1 OFF OFF }
add_pst_state PM5 -pst top_pst -state { HV2 LV2 LV4 }
add_pst_state PM6 -pst top_pst -state { HV2 LV2 OFF }
add_pst_state PM7 -pst top_pst -state { HV2 OFF LV4 }
add_pst_state PM8 -pst top_pst -state { HV2 OFF OFF }

```

Example 13-5 Tcl Script Example

```
load_upf example.upf    ## UPF file defined above

create_scenario s1
read_sdc s1.sdc
set_operating_conditions WCCOM lib1.0V
set_voltage -object_list VDD 1.0
set_voltage -object_list VDDA 0.8
set_voltage -object_list VDDDB 0.9
set_scenario_options -scenarios s1 -setup false -hold false \
-leakage_power true

create_scenario s2
read_sdc s2.sdc
set_operating_conditions BCCOM lib1.1V
set_voltage -object_list VDD 1.1
set_voltage -object_list VDDA 0.6
set_voltage -object_list VDDDB 0.7
set_scenario_options -scenarios s2 -setup false -hold false \
-leakage_power true

compile_ultra -scan -gate_clock
```

Note:

The UPF file is not scenario-specific. As a result, the UPF file must contain port state definitions and power state tables for all the scenarios.

You use the `load_upf` command to read the UPF script shown in [Example 13-4](#).

A

Lower Domain Boundary Support

This appendix describes the lower domain boundary feature in detail. This appendix contains the following sections:

- [Introduction](#)
- [Enabling the Lower Domain Boundary Feature](#)
- [Changes to the Application of the Isolation and Level-Shifter Strategies](#)
- [Impact on Hierarchical Flow](#)

Introduction

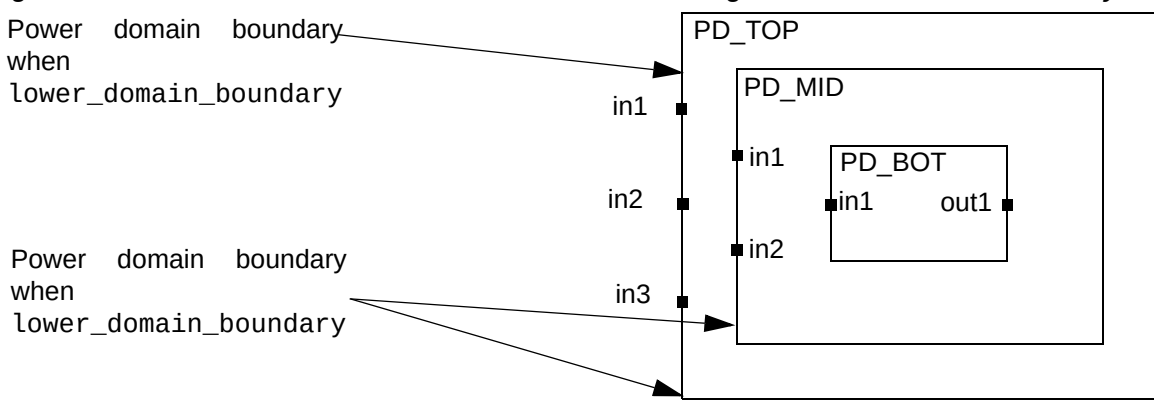
By default, the Power Compiler tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801-2009 (UPF) standard, the tool can consider a power domain boundary to include the boundary of another domain contained in it. This feature enables you to specify the elements on the lower domain boundary for level-shifter and isolation strategy definition, which gives you additional flexibility in selecting the location of the power management cells.

Enabling the Lower Domain Boundary Feature

For the Power Compiler tool to extend the definition of the power domain boundary to the boundary of another power domain contained in it, set the `lower_domain_boundary` attribute to `true`, as shown in the following example:

```
set_design_attributes -elements {.} -attribute lower_domain_boundary true
```

Figure A-1 Definition of Power Domain Boundaries Using the lower_domain_boundary Attribute



In [Figure A-1](#), by default, the tool considers only in1, in2, and in3 ports of the PD_TOP domain as the domain boundary.

When the `lower_domain_boundary` attribute is set to `true`, the tool considers the in1, in2, in3, MID/in1, and MID/in2 ports as the power domain boundary. However, the boundary does not extend to the interface of the BOT design or the PD_BOT power domain.

You must specify the `lower_domain_boundary` attribute on a scope before creating any power domain at the scope or below the scope.

Note that you cannot enable the lower-domain boundary feature at any level below a design in which the feature is disabled.

Changes to the Application of the Isolation and Level-Shifter Strategies

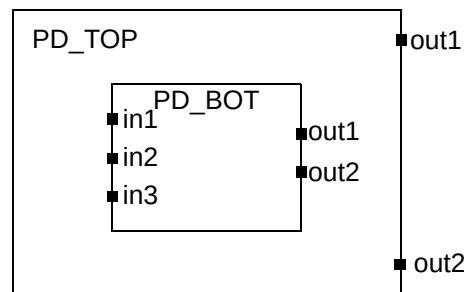
When you set the `lower_domain_boundary` attribute to `true`, the elements considered by the tool for isolation and level-shifting, and the options supported for defining the strategy are as follows:

- The isolation and the level-shifter strategy that you specify applies to the pins of the lower domain boundary.
- The tool does not support the `set_level_shifter -location parent` command.
- The tool does not support the `set_isolation_control -location parent` and `set_isolation_control -location fanout` commands.

When you define the isolation and level-shifter strategies using the `-applies_to` option with the `set_isolation` and `set_level_shifter` commands, the strategies apply to the pins on the lower domain boundary of the power domain. For example, in the nested power domain shown in [Figure A-2](#), the `out1` and `out2` output pins of the `PD_BOT` power domain are input pins for the strategies defined in the `PD_TOP` power domain. Similarly, `in1`, `in2`, and `in3` input pins of the `PD_BOT` power domain are the output pins for the strategies defined in the `PD_TOP` power domain.

So, the isolation and level-shifter strategies that apply to the input pins of the power domain also apply to the output pins of the root cell of the power domain nested inside the power domain. Similarly, the isolation and level-shifter strategies that apply to the output pins of the power domain also apply to the input pins of the root cells of the power domains nested inside the domain.

Figure A-2 Pins Considered for Isolation in Nested Power Domains



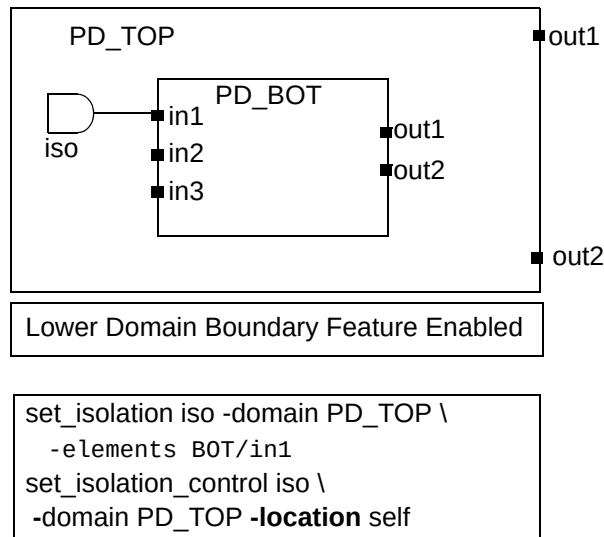
In the following example, the `out_iso` strategy defined for the `PD_TOP` power domain applies to the `BOT/in1`, `BOT/in2`, and `BOT/in3` pins, which are the lower domain boundary pins of the `PD_TOP` power domain.

```
dc_shell> set_scope /
dc_shell> create_power_domain PD_TOP -include_scope
dc_shell> set_design_attributes -elements {.} \
    -attribute lower_domain_boundary true
dc_shell> set_isolation out_iso -domain PD_TOP -applies_to outputs
```

When you enable the lower domain boundary feature, the tool does not support specifying the `-location parent` option for the isolation and the level-shifter strategies. If you require the isolation or level-shifter cell in the parent domain, define the isolation or level-shifter strategy in the parent power domain, as the domain boundary includes the boundary of a lower domain. This eliminates the need to specify `-location parent` option.

The example in [Figure A-3](#) defines an isolation strategy in the `PD_TOP` power domain to isolate the `BOT/in1` pin. When you enable the lower domain boundary feature, you can put the isolation cell in the parent domain by defining the isolation strategy on the `PD_TOP` power domain by using the `-location self` option.

Figure A-3 Isolation Strategy After Enabling Lower Domain Boundary Support



Specifying Design Instances Using Wildcard Characters

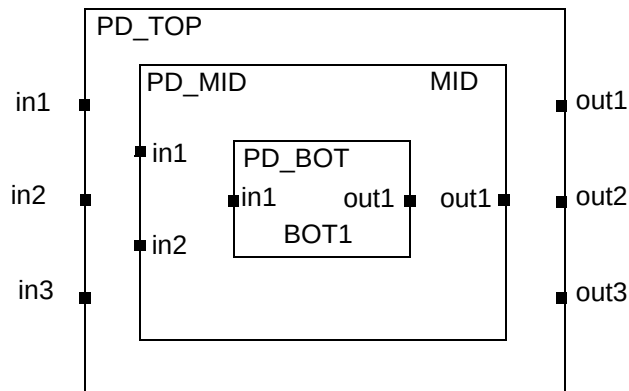
When you use the wildcard characters (`*` or `?`) with the `-elements` option, the tool searches for matching ports, pins, or design instances in the current level of hierarchy and applies the isolation strategy to the elements identified as the boundaries of the specified power domain.

In [Example A-1](#), the asterisk (*) wildcard character is specified in the `-elements` option of the `set_isolation` command for the `tiso` isolation strategy. The tool applies the `tiso` isolation strategy to the LowConn (hierarchically lower) side of the ports and pins of the PD_TOP power domain and HighConn (hierarchically higher) side of the ports and pins of the PD_MID power domain in [Figure A-4](#).

Example A-1 Instance Support Using Asterisk (*) Wildcard Character

```
dc_shell> set_scope /
dc_shell> set_design_attributes -elements {.} \
    -attribute lower_domain_boundary true
dc_shell> create_power_domain PD_TOP
dc_shell> create_power_domain PD_MID -elements {MID}
dc_shell> create_power_domain PD_BOT -elements {MID/BOT1}
dc_shell> set_isolation tiso -domain PD_TOP -elements {*} \
    -applies_to outputs
```

Figure A-4 Design With Nested Power Domains



Specifying Design Instances Using SystemVerilog Elements

When using the `-elements` option with certain UPF commands, you can reference SystemVerilog vector and structure elements using their RTL vector or structure name. To enable this feature, run the following command:

```
set_upf_query_options -bus_struct_mode true
```

Setting the `-bus_struct_mode` option affects the `set_retention`, `set_isolation`, and `map_retention_cell` commands. To use this feature, you must have the following set before reading the UPF:

- `bus_naming_style = "%s[%d]"` (default)
- `bus_dimension_separator_style = "]"` (default)
- `hdl_enable_upf_compatible_naming true`

This option should only be enabled when loading the initial RTL UPF and then should be disabled after reading the RTL, as shown:

```
set_upf_query_options -bus_struct_mode true
load_upf block1.upf
set_upf_query_options -bus_struct_mode false
```

Note that the `-bus_struct_mode` option should be `false` (the default) except when reading in the RTL UPF before any netlist editing happens. Otherwise, any vector or structure references might return incorrect results.

Filtering the Design Elements Using the `-applies_to` Option

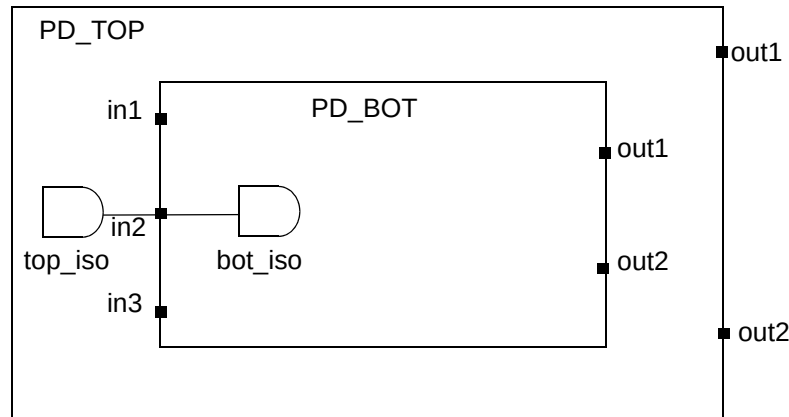
When you specify the `-applies_to` option with the `-elements` option of the `set_isolation` command, the tool filters the elements using the following guidelines:

- The `-applies_to` option is applied relative to the power domain where the isolation strategy is defined.
- If you enable the lower domain boundary feature,
 - For the LowConn side of the port or pin on the interface of the power domain, the tool selects the ports or pins whose direction matches the direction specified with the `-applies_to` option.
 - For the HighConn side of the port or pin on the interface of the power domain, the tool selects the port or pin whose direction is opposite to the direction specified with the `-applies_to` option.

Insertion of Back-to-Back Isolation and Level-Shifter Cells

The lower domain boundary feature enables the insertion of similar power-management cells on either side of the power-domain boundary. For example, insertion of two isolation or level-shifter cells on the same power domain boundary; one cell inside the power domain and the other in the surrounding power domain, as shown in [Figure A-5](#).

Figure A-5 Back-to-Back Isolation Cells



The following example script defines the `top_iso` and `bot_iso` isolation strategies for the isolation cells shown in [Figure A-5](#). The `top_iso` strategy is applicable to BOT/in1 pin, because the pin is on the lower domain boundary of the PD_TOP power domain.

```
dc_shell> set_isolation top_iso -domain PD_TOP -elements BOT/in1
dc_shell> set_isolation_control top_iso -domain PD_TOP -location self
```

```
dc_shell> set_isolation bot_iso -domain PD_BOT -elements BOT/in1
dc_shell> set_isolation_control bot_iso -domain PD_BOT -location self
```

Impact on Hierarchical Flow

When you enable the lower domain boundary feature, the hierarchical flow is simplified in both the top-down and bottom-up flows. You do not need to define the isolation strategies using the `-location parent` option in the block-level design because the equivalent strategies can be defined in the parent domain by including the lower domain boundary elements.

When you specify design instances with the isolation strategy, during characterization of the design instances, the tool characterizes the applicable instance-specific strategies. If the design of the characterized design instance is removed, the upper-domain strategies applied on the design instance is retained in the UPF design of the top-level block.

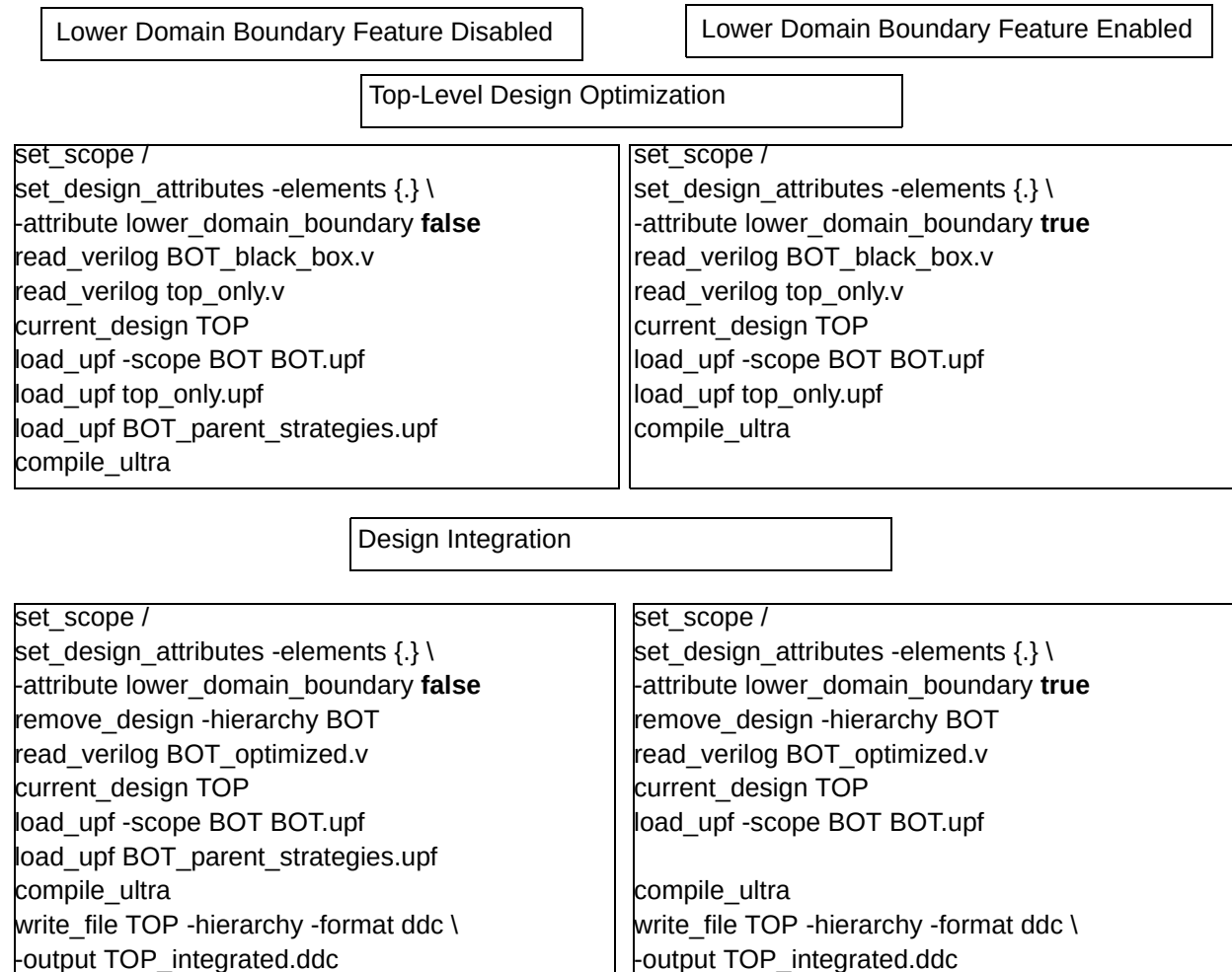
During integration of the implemented blocks, the tool matches the instance-based strategies with their corresponding strategies that are present in the top-level design and performs domain merging.

Bottom-Up Flow

When the lower domain boundary is not enabled, to optimize the top-level design, you follow these steps:

1. Load the netlist and the UPF file for the top-level design
2. Load the block-level design and the UPF file for the block-level design
3. Load the UPF file for block-level design, that contains only the isolation and level-shifter strategies defined using the `-location parent` option
4. Optimize the top-level design

During design integration, when you replace the block-level design by the optimized block-level design, the strategies defined using `-location parent` option are removed. You must reload the UPF file for the block-level design that contains only the strategies that are defined using `-location parent` option.

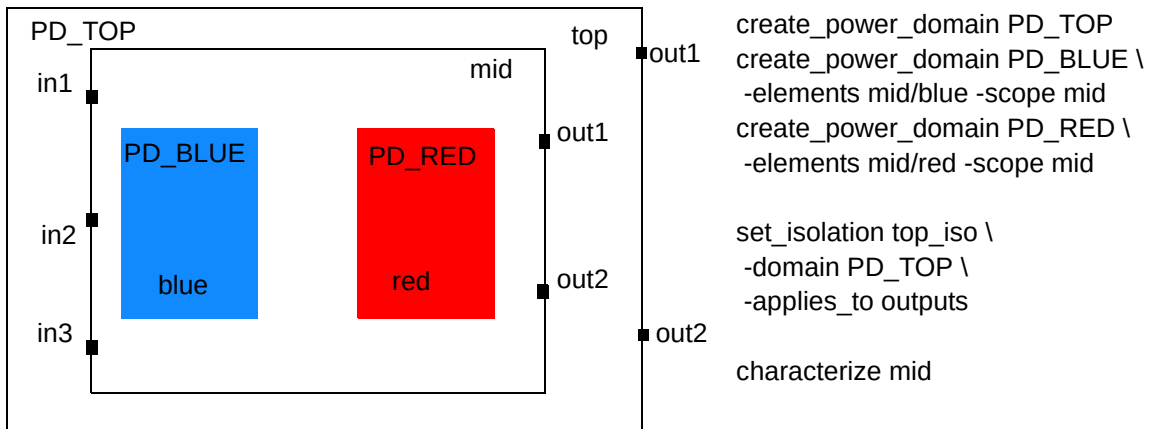
Figure A-6 Bottom-Up Flow

As shown in [Figure A-6](#), when you enable the lower domain boundary feature, during top-level design optimization and design integration, you do not need to load a separate UPF file that contains only the strategies defined using the `-location parent` option.

Top-Down Flow

When you enable the lower domain boundary feature, you can define the isolation strategy on the lower boundary of the top domain. In [Figure A-7](#), the `top_iso` isolation strategy is characterized to the mid block because the `PD_BLUE` and `PD_RED` power domains are lower domain boundaries of the `PD_TOP` power domain. The `top_iso` isolation strategy applies to the `PD_BLUE` and `PD_RED` power domains after characterization. Also, the tool creates an additional isolation strategy with the `-no_isolation` option in the mid power domain to avoid isolating the top-level ports of the mid power domain.

Figure A-7 Top-Down Hierarchical Flow When Lower Domain Boundary is Enabled



Characterization of the Related Supply

In the top-down flow, the characterization of the related supply of a pin is based on the strategy that applies to the pin from the lower-boundary as well as the upper-boundary of the power domain.

- When multiple strategies apply to a pin, from the lower and upper-domain boundaries, the tool derives the related supply for the pin based on the strategies applicable from the upper-domain boundary.
- When an input or output pin drives the data-pin of an isolation cell, the supply of the driver is the related supply of the input or output pin.

B

Integrated Clock-Gating Cell Example

This appendix contains an example .lib description of an integrated clock-gating cell and some schematic examples of rising (positive) and falling (negative) edge integrated clock-gating cells.

This appendix contains the following sections:

- [Library Description](#)
- [Example Schematics](#)

Library Description

[Example B-1](#) is a description of an integrated clock-gating cell that demonstrates the following features:

- The `clock_gating_integrated_cell` attribute
- Appropriate clock-gating attributes on three pins
- Setup and hold arc on enable pin (EN) with respect to the clock pin (CP)
- Combinational arcs from enable pin (EN) and clock pin (CP) to the output pin (Z)
- State table and state function on the output pin (Z)
- Internal power table

Example B-1 HDL Description, Integrated Clock-Gating Cell

```
cell(CGLP) {
  area : 1;
  clock_gating_integrated_cell : "latch_posedge";
  dont_use : true;
  statetable(" CP EN ", "IQ ") {
    table : " L  L  : - : L ,\
              L  H  : - : H ,\
              H  -  : - : N ";
  }
  pin(IQ) {
    direction : internal;
    internal_node : "IQ";
  }
  pin(EN) {
    direction : input;
    capacitance : 0.017997;
    clock_gate_enable_pin : true;
    timing() {
      timing_type : setup_rising;
      intrinsic_rise : 0.4;
      intrinsic_fall : 0.4;
      related_pin : "CP";
    }
    timing() {
      timing_type : hold_rising;
      intrinsic_rise : 0.4;
      intrinsic_fall : 0.4;
      related_pin : "CP";
    }
  }
  pin(CP) {
    direction : input;
    capacitance : 0.031419;
  }
}
```

```

        clock_gate_clock_pin : true;
        min_pulse_width_low  : 0.319;
    }
    pin(Z) {
        direction : output;
        state_function : "CP * IQ";
        max_capacitance : 0.500;
        max_fanout : 8
        clock_gate_out_pin : true;
        timing() {
            timing_sense : positive_unate;
            intrinsic_rise : 0.48;
            intrinsic_fall : 0.77;
            rise_resistance : 0.1443;
            fall_resistance : 0.0523;
            rise_resistance : 0.1443;
            fall_resistance : 0.0523;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "CP";
        }
        timing() {
            timing_sense : positive_unate;
            intrinsic_rise : 0.22;
            intrinsic_fall : 0.42;
            rise_resistance : 0.1443;
            fall_resistance : 0.0523;
            slope_rise : 0.0;
            slope_fall : 0.0;
            related_pin : "EN";
        }
        internal_power () {
            rise_power(li4X3){
                index_1("0.0150, 0.0400, 0.1050, 0.3550");
                index_2("0.050, 0.451, 1.501");
                values("0.141, 0.148, 0.256",\
                    "0.162, 0.145, 0.234",\
                    "0.192, 0.200, 0.284",\
                    "0.199, 0.219, 0.297");
            }
            fall_power(li4X3){
                index_1("0.0150, 0.0400, 0.1050, 0.3550");
                index_2("0.050, 0.451, 1.500");
                values("0.117, 0.144, 0.246",\
                    "0.133, 0.151, 0.238",\
                    "0.151, 0.186, 0.279",\
                    "0.160, 0.190, 0.217");
            }
            related_pin : "CP EN" ;
        }
    }
}

```

When creating your model, examine whether it includes all the `clock_gate` attributes on both the cell and on the pins. Some of the Power Compiler commands require these attributes to recognize the functionality of the cell. DFT Compiler does not recognize this cell. If these attributes are not included, an error message displays. Include the following attributes in your model:

- `clock_gating_integrated_cell`
- `clock_gate_test_pin`
- `clock_gate_enable_pin`
- `clock_gate_out_pin`
- `clock_gate_clock_pin`

Library Compiler can interpret the functionality of the integrated clock-gating cell directly from the state table and state function. The following example shows the `clock_gating_integrated_cell` attribute with a generic value:

```
cell(CGLP) {
  area : 1;
  clock_gating_integrated_cell : "generic";
  dont_use : true;
  statetable(" CP EN ", "IQ ") {
    table : " L L : - : L ,\
L H : - : H ,\
H - : - : N ";
  }
  pin(IQ) {
    direction : internal;
    internal_node : "IQ";
  }
  ...
  pin(Z) {
    direction : output;
    state_function : "CP * IQ";
    max_capacitance : 0.500;
    max_fanout : 8;
    clock_gate_out_pin : true;
    timing() {
    ...
  }
}
```

Example Schematics

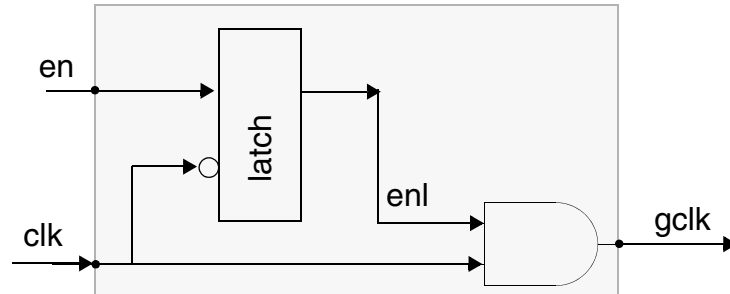
This section contains example schematics of latch-based and latch-free clock-gating styles for rising- and falling-edge-triggered logic. These are a subset of integrated clock-gating cells supported by Power Compiler.

Rising-Edge Latch-Based Integrated Cells

The following integrated cells are latch-based. The rising-edge latch-free integrated cells are described in the following section.

[Figure B-1](#) displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs.

Figure B-1 Rising-Edge Latch-Based Integrated Cell (latch_posedge)



[Figure B-2](#) displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-2 Rising-Edge Latch-Based Integrated Cell With Pre-Control (latch_posedge_precontrol)

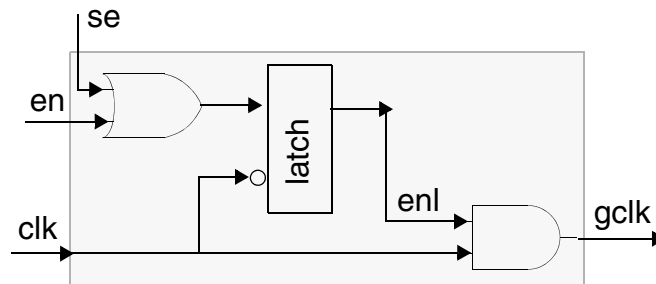


Figure B-3 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-3 Rising-Edge Latch-Based Integrated Cell With Post-Control
(*latch_posedge_postcontrol*)

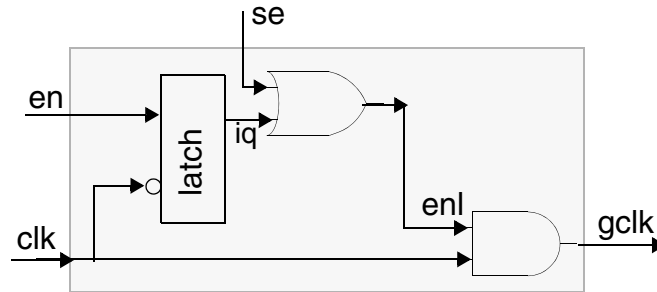


Figure B-4 Rising Edge Latch Based Integrated Cell With Post-Control Observable Point
(*latch_posedge_postcontrol*)

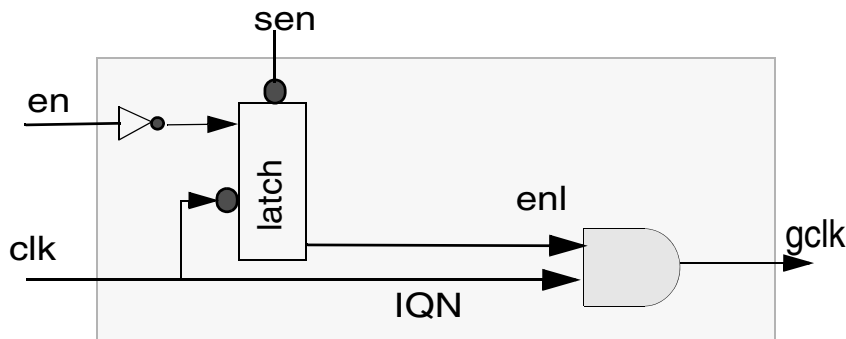


Figure B-5 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-5 Rising-Edge Latch-Based Integrated Cell With Pre-Control Observable Point
(*latch_posedge_precontrol_obs*)

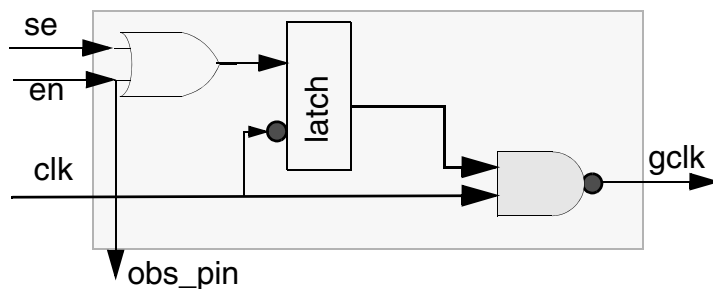
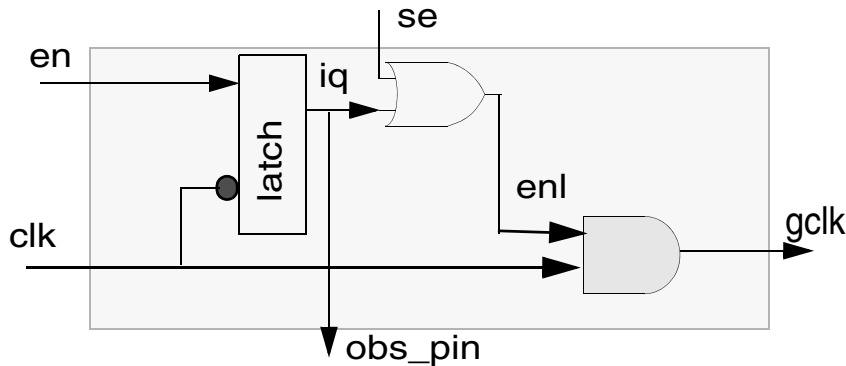


Figure B-6 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-6 Rising-Edge Latch-Based Integrated Cell With Post-Control Observable Point (*latch_posedge_postcontrol_obs*)



Rising-Edge Latch-Free Integrated Cells

The following integrated cells are latch-free. The rising-edge latch-based integrated cells were described in the previous section.

Figure B-7 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs.

Figure B-7 Rising-Edge Latch-Free Integrated Cell (*none_posedge*)

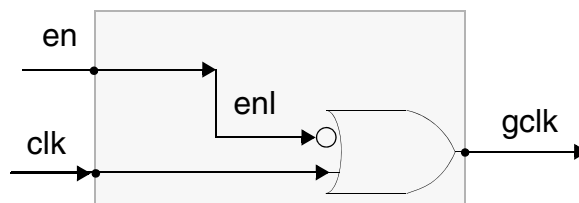


Figure B-8 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-8 Rising-Edge Latch-Free Integrated Cell With Control (*none_posedge_control*)

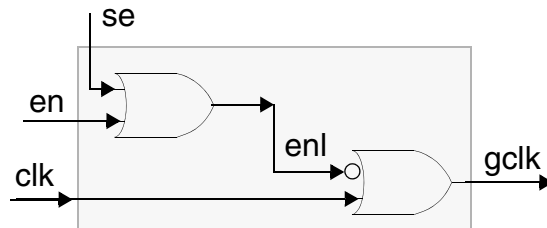
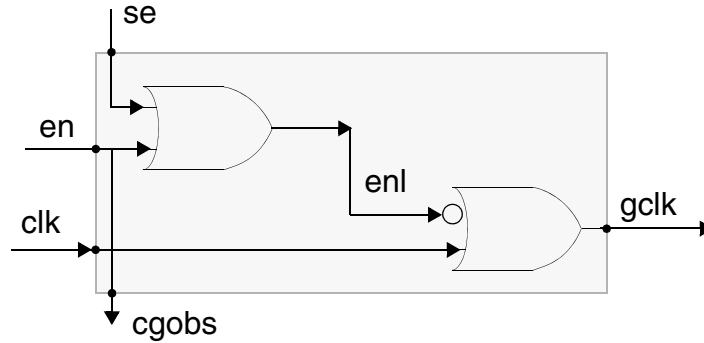


Figure B-9 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from rising-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-9 Rising-Edge Latch-Free Integrated Cell With Control Observable Point (none_posedge_control_obs)



Falling Edge Latch-Based Integrated Cells

The following integrated cells are latch-based. The falling-edge latch-free integrated cells are described in the following section.

Figure B-10 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs.

Figure B-10 Falling-Edge Latch-Based Integrated Cell (latch_negedge)

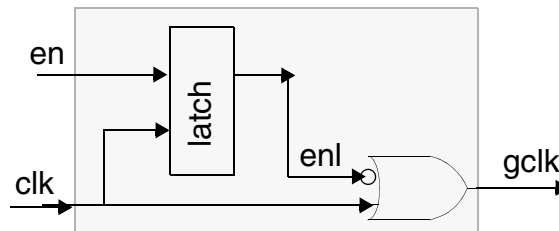


Figure B-11 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-11 Falling-Edge Latch-Based Integrated Cell With Pre-Control Observable Point (*latch_negedge_precontrol*)

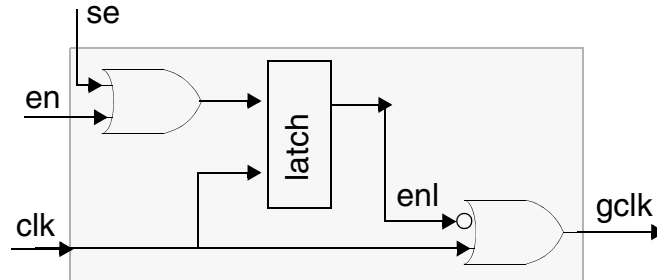


Figure B-12 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-12 Falling-Edge Latch-Based Integrated Cell With Post-Control Observable Point (*latch_negedge_postcontrol*)

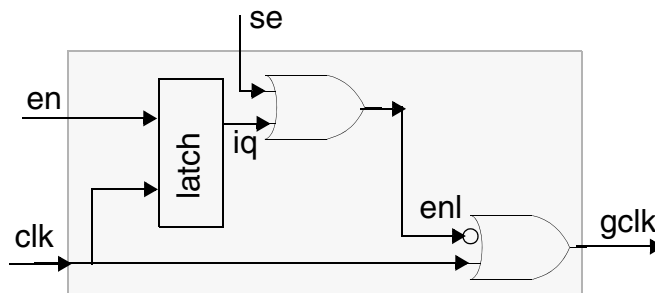


Figure B-13 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-13 Falling-Edge Latch-Based Integrated Cell With Pre-Control Observable Point (*latch_negedge_precontrol_obs*)

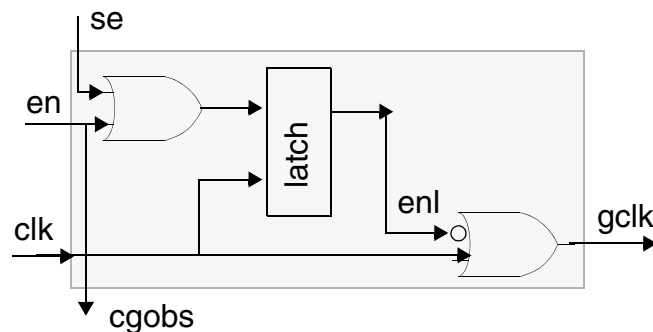
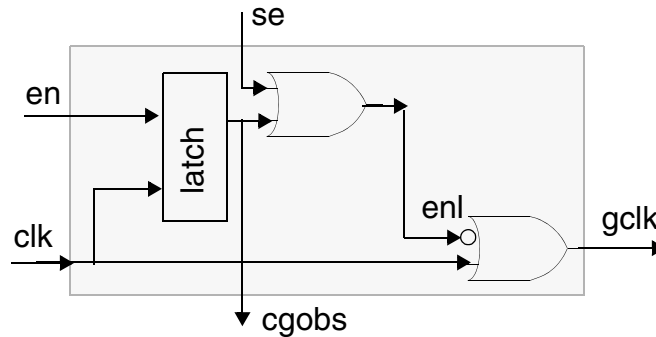


Figure B-14 displays an integrated cell using a latch-based gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-14 Falling-Edge Latch-Based Integrated Cell With Post-Control Observable Point (*latch_negedge_postcontrol_obs*)



Falling-Edge Latch-Free Integrated Cells

The following integrated cells are latch-free. The falling-edge latch-based integrated cells were described in the previous section.

Figure B-15 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs.

Figure B-15 Falling-Edge Latch-Free Integrated Cell (*none_negedge*)

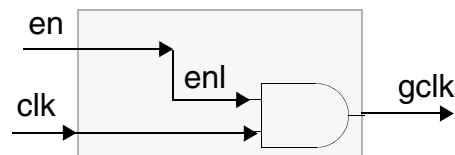


Figure B-16 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable).

Figure B-16 Falling-Edge Latch-Free Integrated Cell With Control (*none_negedge_control*)

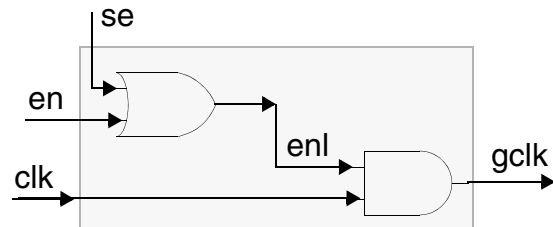
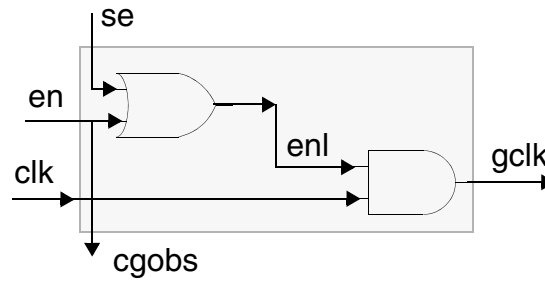


Figure B-17 displays an integrated cell using a latch-free gating style, appropriate for registers inferred from falling-edge-triggered HDL constructs. The integrated cell contains test logic (scan enable) and observable point (cgobs).

Figure B-17 Falling-Edge Latch-Free Integrated Cell With Control Observable Point (*none_negedge_control_obs*)



C

Attributes for Querying and Filtering

This appendix describes derived Power Compiler attributes that you can use in scripts to view and filter design objects related to clock gating for power optimization.

The derived attributes described in this appendix are read-only properties that Power Compiler automatically assigns to designs, cell, and pins based on other attributes or the netlist configuration.

At times, you might want to view and use design objects according to their attributes. For example, you might want to filter for cells that are integrated clock gates (the `is_icg` attribute). Or, your queries might be required for back end processes such as clock tree synthesis in which fanout considerations have priority.

This appendix contains the following sections:

- [Derived Attribute Lists](#)
- [Usage Examples](#)

Derived Attribute Lists

You can query for the following derived attributes assigned by Power Compiler. Specify `man power_attributes` in `dc_shell` to view a list of these attributes. [Table C-1](#) and [Table C-2](#) show the derived attributes for designs and cells, respectively.

Table C-1 Derived Attributes for Designs

Name	Type	Description
<code>is_clock_gating_design</code>	Boolean	true if the design is a clock-gating design
<code>is_clock_gating_observability_design</code>	Boolean	true if the design is a clock-gating observable design

Table C-2 Derived Attributes for Cells

Name	Type	Description
<code>is_clock_gate</code>	Boolean	true if the cell is a clock gate
<code>is_icg</code>	Boolean	true if the cell is an integrated clock gate
<code>is_gicg</code>	Boolean	true if the cell is a generic integrated clock gate
<code>is_latch_based_clock_gate</code>	Boolean	true if the cell is a latch-based clock-gating cell
<code>is_latch_free_clock_gate</code>	Boolean	true if the cell is a latch-free clock-gating cell
<code>is_positive_edge_clock_gate</code>	Boolean	true if the cell is a positive edge clock gate
<code>is_negative_edge_clock_gate</code>	Boolean	true if the cell is a negative edge clock gate
<code>clock_gate_has_precontrol</code>	Boolean	true if the cell is a clock gate with (pre-latch) control point
<code>clock_gate_has_postcontrol</code>	Boolean	true if the cell is a clock gate with (post-latch) control point
<code>clock_gate_has_observation</code>	Boolean	true if the cell is a clock gate with observation point
<code>is_clock_gated</code>	Boolean	true if the cell is a clock-gated register or clock gate

Table C-2 Derived Attributes for Cells (Continued)

Name	Type	Description
clock_gating_depth	integer	number of clock gates on the clock path to this cell; -1 if not a clock gate or register
clock_gate_level	integer	position in a multistage clock tree: number of clock gates on the longest branch in the fan out of this cell; -1 if not a clock gate
clock_gate_fanout	integer	number of registers and clock gates in the direct fan out of the clock gate; -1 if not a clock gate
clock_gate_register_fanout	integer	number of registers in the direct fan out of the clock gate; -1 if not a clock gate
clock_gate_multi_stage_fanout	integer	number of clock gates in the direct fan out of the clock gate; -1 if not a clock gate
clock_gate_transitive_register_fanout	integer	number of registers in the transitive fan out of the clock gate; -1 if not a clock gate
clock_gate_module_fanout	integer	number of modules in the local fan out of the clock gate; -1 if not a clock gate

For hierarchical clock-gating cells, the derived clock-gating attributes only work when applied to the hierarchical clock-gate wrapper. If you apply an attribute to the leaf cell of a discrete clock gate or a leaf integrated clock gate, the attribute returns false for Boolean attributes, -1 for integer attributes, or an empty string for string attributes. The only exception to this rule is the `is_icg` attribute; this attribute is true when applied to a leaf integrated clock gate contained within a hierarchical clock gate wrapper but false when applied to that wrapper. This behavior allows you to recognize the actual integrated clock-gating cell, not the hierarchical wrapper.

Table C-3 Derived Attributes for Pins

Name	Type	Description
is_clock_gate_enable_pin	Boolean	true if the pin is a clock-gate enable input
is_clock_gate_clock_pin	Boolean	true if the pin is a clock-gate clock input
is_clock_gate_output_pin	Boolean	true if the pin is a clock-gate gated-clock output
is_clock_gate_test_pin	Boolean	true if the pin is a clock-gate scan-enable or test-mode input

Table C-3 Derived Attributes for Pins (Continued)

Name	Type	Description
is_clock_gate_observation_pin	Boolean	true if the pin is a clock-gate observation point

Usage Examples

You can query the attributes described in the previous section using the `get_attribute`, `get_designs`, `get_cells`, `get_pins`, and `all_clock_gates` commands. You can also use these commands with the `-filter` option.

The following examples show how the attributes might appear in scripts.

To gather all the clock gates specific to a clock "clk":

```
all_clock_gates -clock [ get_clocks clk]
```

The `all_clock_gates` command creates a collection of clock-gating cells or pins that satisfy the parameters you set. Additional options allow you to filter for enable, clock, and gated-clock pins; scan_enable or test_mode pins; and observation pins. For more information, see the man page.

To filter out the multistage clock-gating cell associated with the clock "clk":

```
set multi_stage_cg [filter [all_clock_gates -clock \
    [get_clocks clk]] \ "@clock_gate_level >0" ]
```

To retrieve the number of fan outs of a clock-gating cell:

```
get_attribute [ get_cells top/clock_gate_1 ] \
    clock_gate_fanout
```

To gather a collection of clock-gating cells with pre-latch control point and a fanout greater than four:

```
set CG_collection [filter [all_clock_gates] \
    "@clock_gate_has_precontrol== \
    true && @clock_gate_fanout > 4"]
```

To gather a collection of clock-gating designs (the wrapper design where the clock-gating cells reside):

```
set CG_designs [get_designs -filter \
    "@is_clock_gating_design==true"]
```

D

Power Compiler Command and Variable Reference

This appendix lists the Power Compiler tool commands and variables.

Getting Help

Power Compiler provides various forms of online help.

- The `help` command provides quick help for one or more commands or procedures.
- The `man` command displays the man page.

You can use a wildcard pattern as the argument for the `help` command. The wildcard characters are

- `*` matches *n* characters.
- `?` matches exactly one character.

Accessing Help

Use this command to list all commands by the function group:

```
dc_shell> help
```

Use this command to display all commands that end with the word `clock`:

```
dc_shell> help *clock
```

Use this command to get syntax help for one or more commands:

```
dc_shell> help -verbose command_name_pattern
```

Use this command to get syntax help for a specific command:

```
dc_shell> command_name -help
```

Man Page Viewing Instructions

The following sections describe how to set up your environment and the syntax required to view man pages.

Viewing Man Pages in SolvNet

You can view the man pages in HTML or PDF format on the Synopsys SolvNet® online support site. To view the man pages, go to:

<https://solvnet.synopsys.com/DocsOnWeb>

Click Man Pages and Error Messages on the right side of the page.

Setting Up the UNIX Environment

Edit your `.cshrc` file to contain these lines:

```
setenv SYN_MAN_DIR synopsys_root/doc/syn/man
setenv MANPATH ${MANPATH}:${SYN_MAN_DIR}
```

`SYN_MAN_DIR` is a variable that contains the path to the man page directories, and *synopsys_root* represents the specific path to the Synopsys software directory at your site.

Viewing Man Pages From UNIX

To view command or variable man pages from UNIX, enter the following command:

```
% man command_or_variable_name
```

It is not possible to view the man pages for error, warning, and information messages from UNIX.

Viewing Man Pages From dc_shell

Command:

```
dc_shell> man command_name
```

Variable:

```
dc_shell> man variable_name
```

Error, warning, or information message:

```
dc_shell> man message_id
```

Power Compiler Commands

Invoke these commands from within the Power Compiler tool. For more information about these commands, see the man pages.

add_power_state

Adds state information to a supply set.

```

status add_power_state
[-supply | -group] object_name
[-simstate NORMAL | CORRUPT_ON_ACTIVITY | CORRUPT | NOT_NORMAL]
[-update]
[-state {state_name [-supply_expr {boolean_function}]
                    [-logic_expr {boolean_function}]
                    [-simstate NORMAL | CORRUPT_ON_ACTIVITY | CORRUPT |
                     NOT NORMAL]
                    [-illegal]]}*

```

add_pst_state

Defines the states of each of the supply nets for one possible state of the design.

```

status add_pst_state
state_name
-pst table_name
-state supply_states

```

all_clock_gates

Returns a collection of clock-gating cells or pins in the current design.

```

collection all_clock_gates
[-no_hierarchy]
[-clock clock_name]
[-cells]
[-enable_pins]
[-clock_pins]
[-output_pins]
[-test_pins]
[-observation_pins]

```

all_isolation_cells

Returns a collection of isolation cells available in the design.

```

collection all_isolation_cells

```

all_level_shifters

Returns a collection of level-shifter cells available in the design.

```

collection all_level_shifters [-type els | simple]

```

all_self_gates

Returns a collection of self-gating cells or pins in the current design. The command is supported only in topographical mode.

```
collection all_self_gates
[-no_hierarchy]
[-clock clock_name]
[-cells]
[-enable_pins]
[-clock_pins]
[-output_pins]
[-test_pins]
```

all_upf_repeater_cells

Returns a collection of repeater cells available in the design that have the UPF repeater-supply port attribute.

```
collection all_upf_repeater_cells
```

analyze_library

Provides a library cell analysis based on the specified parameters

```
status analyze_library
[-multi_vth]
[-ignore_dont_use_attribute]
[-graph file_name]
[list of libraries]
```

analyze_dw_power

Reports the DesignWare delay and power contribution.

```
integer analyze_dw_power
[-nosplit]
[-hierarchy]
[-sort slack | dyn_pwr | lkg_pwr]
[-sort_ascending]
```

analyze_mv_design

Analyzes multivoltage design connections.

```
status analyze_mv_design
```

```
[-level_shifter | -always_on]
[-from_pin from_pin_list]
[-to_pin to_pin_list]
[-net target_net]
[-verbose]
```

apply_clock_gate_latency

Annotates the clock latencies on the existing clock-gating cells based on the settings previously specified using the `set_clock_gate_latency` command.

```
status apply_clock_gate_latency
```

associate_supply_set

Associates a supply set handle to another supply set handle or supply set reference.

```
status associate_supply_set
supply_set_name
-handle supply_set_handle
```

characterize

Captures information about the environment of specific cell instances and assigns the information as attributes on the design to which the cells are linked.

```
status characterize
cell_list
[-no_timing]
[-constraints]
[-connections]
[-power]
[-verbose]
```

check_level_shifters

Checks the design for all existing level shifters and nets against the specified level-shifter strategy and threshold.

```
status check_level_shifters [-verbose]
```

check_mv_design

Checks for violations in a multivoltage design.

```
status check_mv_design
[-verbose]
[-isolation]
[-target_library_subset]
[-opcond_mismatches]
[-connection_rules]
[-level_shifters]
[-power_nets]
[-clock_gating_style]
[-max_messages message_count]
[-output output_file_name]
```

compile

Performs logic-level and gate-level synthesis and optimization on the current design.

```
status compile
[-no_map]
[-map_effort medium | high]
[-area_effort none | low | medium | high]
[-incremental_mapping]
[-exact_map]
[-ungroup_all]
[-boundary_optimization]
[-auto_ungroup area | delay]
[-no_design_rule | -only_design_rule | -only_hold_time]
[-scan]
[-top]
[-power_effort none | low | medium | high]
[-gate_clock]
```

compile_ultra

Performs a high-effort compile on the current design for better quality of results (QoR).

```
status compile_ultra
[-incremental]
[-scan]
[-exact_map]
[-no_autoungroup]
[-no_seq_output_inversion]
[-no_boundary_optimization]
[-no_design_rule]
[-only_design_rule]
[-timing_high_effort_script]
[-area_high_effort_script]
[-top]
[-retime]
[-gate_clock]
[-self_gating]
[-check_only]
[-congestion]
[-spg]
[-no_auto_layer_optimization]
```

connect_logic_net

Connects a logic net to logic ports. Returns the status of the command: 1 if successful and 0 when unsuccessful.

```
status connect_logic_net net_name [-ports port_list]
```

connect_supply_net

Connects the supply net to the specified supply ports and pins.

```
status connect_supply_net supply_net_name
-ports list
[-vct vct_name]
```

convert_pg

Converts RTL power and ground information extracted from the RTL into the UPF format.

```
status convert_pg
[-net_creation_style domain_dependent | domain_independent]
```

create_logic_net

Defines a logic net.

```
status create_logic_net net_name
```

create_logic_port

Defines a logic port.

```
string create_logic_port port_name [-direction in | out | inout]
```

create_power_domain

Creates a power domain, which provides a power supply distribution network.

```
string create_power_domain  
domain_name  
[-elements cells]  
[-include_scope]  
[-scope instance_name]  
[-supply {supply_set_handle_name supply_set_name*}]  
[-update]
```

create_power_state_group

Creates a name for a group of related power states.

```
string create_power_state_group group_name
```

create_power_switch

Creates a power switch in the specified power domain.

```
string create_power_switch  
switch_name  
-domain domain_name  
-output_supply_port {port_name supply_net_name}  
-input_supply_port {port_name supply_net_name}  
-control_port {port_name net_name}  
[-ack_port {port_name net_name [{boolean_function}]}]  
[-ack_delay {port_name delay}]  
-on_state {state_name input_supply_port {boolean_function}}  
[-off_state {state_name {boolean_function}}]
```

create_pst

Creates a power state table using a specific order of supply nets.

```
string create_pst table_name -supplies list
```

create_supply_net

Creates a supply net for the specified power domain. The supply net is created in the logic hierarchy at the same scope as the specified power domain.

```
string create_supply_net  
supply_net_name  
[-domain domain_name]  
[-reuse]  
[-resolve unresolved | parallel]
```

create_supply_port

Creates a supply port in the specified power domain. If a power domain is not specified, creates the port in the current scope.

```
string create_supply_port  
supply_port_name  
[-domain domain_name]  
[-direction in | out]
```

create_supply_set

Creates a set of supplies that can be used to define the power network. A supply set is created in the current logic hierarchy.

```
string create_supply_set  
supply_set_name  
[-function {function_name supply_net_name}]*  
[-update]
```

find_objects

Finds logical hierarchy objects within a scope. Returns a list of the found hierarchical names (relative to the active scope); when nothing is found, a null string is returned. This is a UPF query command.

```
list find_objects
scope
-pattern pattern_string
-object_type type_name
[-direction direction_name]
[-transitive transitive_string]
[-non_leaf]
[-leaf_only]
[-exact]
```

get_power_domains

Creates a collection of power domains that match the specified criteria.

```
collection get_power_domains
[-quiet]
[-regex]
[-nocase]
[-filter expression]
[-hierarchical]
[patterns | -of_objects objects]
```

get_power_switches

Creates a collection of power switches that match the specified criteria.

```
collection get_power_switches
[-quiet]
[-regex]
[-nocase]
[-filter expression]
[-hierarchical]
[patterns]
```

get_related_supply_net

Creates a collection of related supply nets of pins.

```
collection get_related_supply_net [pins] [-ground]
```

get_supply_nets

Creates a collection of supply nets that match the specified criteria.

```
collection get_supply_nets  
[-quiet]  
[-regex]  
[-nocase]  
[-filter expression]  
[-hierarchical]  
[patterns]
```

get_supply_ports

Creates a collection of supply ports that match the specified criteria.

```
collection get_supply_ports  
[-quiet]  
[-regex]  
[-nocase]  
[-filter expression]  
[-hierarchical]  
[patterns]
```

generate_mv_constraints

Derives new isolation strategies for the elements whose driving constant value does not match with the isolation clamp value.

```
status generate_mv_constraints  
-align_isolation_clamp_value  
[-output output_file_name]  
[-include_elements]  
[-apply]
```

identify_clock_gating

Identifies clock-gating circuitry inserted by Power Compiler in a structural netlist.

```
status identify_clock_gating  
[-gating_elements cell_collection]
```

infer_switching_activity

Propose and set the switching activity annotation on drivers of special pins of the current design.

```
int infer_switching_activity
[-apply]
[-output file_name]
[-nosplit]
[-verbose]
```

insert_clock_gating

Performs clock gating on an appropriately-prepared GTECH netlist.

```
status insert_clock_gating
[-regular_only]
[-global]
[-no_hier]
```

insert_isolation_cell

Inserts isolation cells on the specified nets, pins, or ports. Isolation cell is a general term that applies to isolation cells and enabled level-shifter cells.

```
status insert_isolation_cell
[-force]
[-verbose]
-enable enable_signal
-object_list objects
-reference lib_cell_name
```

insert_mv_cells

Inserts isolation and level-shifter cells in the design.

```
status insert_mv_cells
[-isolation]
[-level_shifter]
[-all]
[-verbose]
```

lib2saif

Creates a forward-annotation SAIF file for a specified technology library.

```
status lib2saif
[-output file_name]
library
[-lib_pathname lib_path_name]
```

load_upf

Reads a script in the IEEE 1801 Unified Power Format (UPF).

```
status load_upf
upf_file_name
[-scope instance_name]
[-noecho]
[-simulation_only]
[-strict_check true | false]
[-supplemental supf_file_name]
```

map_isolation_cell

Specifies how to map or remap the isolation and enable level-shifter cells belonging to the specified isolation strategy.

```
status map_isolation_cell
isolation_strategy
-domain power_domain
-lib_cells lib_cells
```

map_level_shifter_cell

Specifies that the level-shifter cells belonging to the specified strategy can only be mapped to a subset of the library cells.

```
status map_level_shifter_cell
level_shifter_strategy
-domain power_domain
-lib_cells lib_cells
```

map_power_switch

Defines which power switch library cells to use for the mapping of the given UPF power switch.

```
status map_power_switch
switch_name
-domain domain_name
-lib_cells name_list
```

map_retention_cell

Defines how to map the unmapped sequential cells to retention cells for the specified UPF retention strategy of the power domain.

```
status map_retention_cell
retention_strategy
-domain power_domain
[-lib_cells lib_cells]
[-lib_cell_type lib_cell_type]
[-elements objects]
[-lib_model_name name {-port port_name net_ref}*]
```

merge_saif

Reads a list of SAIF files with their corresponding weights, computes the merged toggle rate and static probability, and annotates the switching activity for the nets, pins, and ports in the current design. The command then generates a merged output SAIF file.

```
status merge_saif
-input_list saif_file_and_weight_list
[-instance_name inst_name]
[-output merged_saif_name]
[-simple_merge]
[-ignore ignore_name]
[-ignore_absolute ig_absolute_name]
[-exclude exclude_file_name]
[-exclude_absolute ex_absolute_file_name]
[-unit_base unit_value]
[-scale scale_value]
[-khrate khrate_value]
[-map_names]
```

propagate_constraints

Propagates timing constraints from lower levels of the design hierarchy to the current design.

```

status propagate_constraints
[-design design_list]
[-all]
[-clocks]
[-disable_timing]
[-dont_apply]
[-false_path]
[-gate_clock]
[-ideal_network]
[-ignore_from_or_to_port_exceptions]
[-ignore_through_port_exceptions]
[-max_delay]
[-min_delay]
[-multicycle_path]
[-operating_conditions]
[-power_supply_data]
[-output file_name]
[-port_isolation]
[-verbose]
[-case_analysis]
[-target_library_subset]
[-opcond_inference]

```

propagate_switching_activity

Forces the propagation of power-switching activity information.

```

status propagate_switching_activity
[-effort low | medium | high]
[-verbose]
[-infer_related_clocks]

```

query_cell_instances

Queries the instances of a cell or module within the active scope. Returns a list of the instances. This is a UPF query command.

```
list query_cell_instances cell_name [-domain domain_name]
```

query_cell_mapped

Queries which cell is mapped to this instance. Returns a cell name. This is a UPF query command.

```
string query_cell_mapped instance_name
```

query_map_power_switch

Returns information about the previous mapping of a library cell to a power switch in the active scope. This is a UPF query command.

```
list query_map_power_switch switch_name [-detailed]
```

query_net_ports

Queries a list of port names that are logically connected to the specified net. If no ports are connected, a null string is returned. This is a UPF query command.

```
list query_net_ports  
net_name  
[-transitive transitive_option]  
[-leaf]
```

query_port_net

Queries the net logically connected to a port. Returns the name of the net connected to the specified port name. If no net is connected, a null string is returned. This is a UPF query command.

```
string query_port_net port_name [-conn connection_option_string]
```

query_port_state

Returns information about the port states that have been previously defined for a specified supply port in the active scope. This is a UPF query command.

```
list query_port_state port_name [-state state_name] [-detailed]
```

query_power_switch

Returns information about a power switch that was previously created in the active scope. This is a UPF query command.

```
list query_port_switch switch_name [-detailed]
```

query_pst

Returns information about the power state tables that have been previously created in the active scope. This is a UPF query command.

```
list query_pst table_name [-detailed]
```

query_pst_state

Returns information about a state in a power state table that was previously created in the active scope. This is a UPF query command.

```
list query_pst_name state_name -pst table_name [-detailed]
```

read_saif

Reads a SAIF file and annotates switching activity information on nets, pins, ports, and cells in the current design.

```
status read_saif
-input file_name
[-instance_name name]
[-target_instance instance]
[-ignore ignore_name]
[-ignore_absolute ig_absolute_name]
[-exclude exclude_file_name]
[-exclude_absolute ex_absolute_file_name]
[-names_file name_changes_log_file]
[-scale scale_value]
[-unit_base unit_value]
[-khrate khrate_value]
[-map_names]
[-auto_map_names]
[-verbose]
```

remove_clock_gating

Directs the compile -incremental_mapping and compile_ultra -incremental commands to remove clock gating from objects clock-gated by Power Compiler.

```
status remove_clock_gating
[-gated_registers gated_register_list]
[-min_bitwidth minsize_value]
[-gating_cells clock_gating_cells_list]
[-all]
[-no_hier]
[-verbose]
[-undo]
```

remove_clock_gating_style

Removes the clock gating style applied on a given set of hierarchical cells or power domains.

```
status remove_clock_gating_style  
[-instances {cell_list}]  
[-power_domains {power_domain_list}]  
[-designs designs]
```

remove_dft_clock_gating_pin

Removes all DFT clock-gating pin specifications for the current design.

```
status remove_dft_clock_gating_pin
```

remove_isolation_cell

Removes the specified isolation cells from the design.

```
status remove_isolation_cell  
[-force]  
-object_list cells
```

remove_level_shifters

Removes all of the level shifters from the design.

```
status remove_level_shifters [-force]
```

remove_power_domain

Removes the specified power domains.

```
status remove_power_domain power_domains | -all
```

remove_upf

Removes the UPF constraints from the design. This command is only supported in `dc_shell`.

```
status remove_upf
```

replace_clock_gates

Replaces manually-inserted clock gates with Power Compiler clock gates.

```
status replace_clock_gates  
[-global]  
[-no_hier]
```

report_clock_gating

Reports the details of clock gating performed by Power Compiler.

```
status report_clock_gating  
[-no_hier]  
[-verbose]  
[-gated]  
[-ungated]  
[-gating_elements]  
[-only cell_list]  
[-nosplit]  
[-physical]  
[-multi_stage]  
[-style]  
[-structure]  
[-enable_conditions]  
[-scenarios scenario_list]
```

report_dft_clock_gating_configuration

Displays the options specified by the `set_dft_clock_gating_configuration` command.

```
status report_dft_clock_gating_configuration
```

report_dft_clock_gating_pin

Displays the specification specified by the `set_dft_clock_gating_pin` command.

```
integer report_dft_clock_gating_pin
```

report_isolation_cell

Displays information about isolation cells in the current scope.

```
status report_isolation_cell
[isolation_cells]
[-domain power_domains]
[-isolation_strategy isolation_strategy_names]
[-ports pins_ports]
[-verbose]
[-nosplit]
```

report_level_shifter

Displays information about level-shifter cells in the current scope.

```
status report_level_shifter
[level_shifter_cells]
[-domain power_domains]
[-verbose]
[-nosplit]
```

report_lib

Displays information about the specified logic library, physical library, or symbol library.

```
status report_lib
[-all]
[-ccs_recv]
[-em]
[-fpga]
[-k_factors]
[-power]
[-power_label]
[-table]
[-full_table]
[-timing]
[-timing_arcs]
[-timing_label]
[-noise]
[-vhdl_name]
[-yield]
[-switch]
[-pg_pin]
[-char]
[-operating_condition]
[-op_cond_name op_cond_name]
[-routing_rule]
[-rwm]
```

```
[-user_defined_data]
[-noise_arcs]
[-jcr]
library_name
cell_list
```

report_mv_library_cells

Displays power management cells available in the target libraries.

```
status report_mv_library_cells
[-level_shifters]
[-isolation_cells]
[-retention_cells]
[-switch_cells]
[-always_on_cells]
[-cell_name master_cell_name]
[-verbose]
```

report_power

Calculates and reports dynamic and static power for the design or instance.

```
status report_power
[-net]
[-cell]
[-only cell_or_net_list]
[-hierarchy]
[-levels level_value]
[-verbose]
[-cumulative]
[-flat]
[-exclude_boundary_nets]
[-include_input_nets]
[-analysis_effort low | medium | high]
[-nworst number]
[-sort_mode mode]
[-histogram [-exclude_leq le_val | -exclude_geq ge_val]]
[-nosplit]
[-scenarios scenario_list]
[-groups group_list]
```

report_power_calculation

Displays the calculation of the internal power for a pin, the leakage power for a cell, or the switching power for a net.

```
status report_power_calculation
pin_cell_or_net_list
[-state_condition boolean_eq_of_pins | default | all]
[-path_sources pin_name | default | all]
[-rise]
[-fall]
[-verbose]
[-nosplit]
```

report_power_domain

Reports information about the specified power domain.

```
status report_power_domain
[-hierarchy]
[-verbose]
[power_domains]
```

report_power_gating

Reports the power-gating style of retention registers in the design.

```
status report_power_gating
[cell_or_design_list]
[-missing]
[-unconnected]
```

report_power_pin_info

Reports the power pin information for technology library cells or leaf cells. The command reports power pin information only for instantiated cells and not the library cells.

```
status report_power_pin_info object_list
```

report_power_switch

Reports all of the specified power switches.

```
status report_power_switch
[power_switch_name]
[-verbose]
```

report_pst

Reports the power states in the current design.

```
status report_pst
[-supplies supply_list]
[-verbose]
[-scope instance_name]
[-derived]
```

report_retention_cell

Displays information about retention cells in the current scope.

```
status report_retention_cell
[retention_cells]
[-domain power_domains]
[-retention_strategy retention_strategy_names]
[-verbose]
```

report_saif

Reports the statistics of switching activity annotation on the current design or instance.

```
status report_saif
[-only cell_or_net_list]
[-hier]
[-missing]
[-annotated_flag]
[-rtl_saif]
```

report_self_gating

Reports information about XOR self-gating performed by Power Compiler. This command is supported only in topographical mode.

```
status report_self_gating
[-ungated]
[-nosplit]
```

report_supply_net

Reports all supply nets in the current scope.

```
status report_supply_net  
[supply_net_name]  
[-include_exception]
```

report_supply_port

Reports information about the supply ports in the current scope.

```
status report_supply_port [supply_port_name]
```

reset_clock_gate_latency

Resets all clock-latency values previously specified for or applied to clock-gating cells.

```
status reset_clock_gate_latency [-clock clock_list]
```

reset_dft_clock_gating_configuration

Resets the DFT clock-gating configuration for the current design.

```
status reset_dft_clock_gating_configuration
```

reset_switching_activity

Removes the toggle rate and static probability attributes, from nets, pins, cells, and ports of the current design.

```
status reset_switching_activity  
[-verbose]  
[object_list]
```

rewire_clock_gating

Changes the clock-gating cell implemented by the tool for a particular gated cell.

```
status rewire_clock_gating  
[-gating_cell new_clock_gating_cell]  
[-gated_objects gated_objects_list]  
[-balance_fanout]  
[-undo]  
[-verbose]
```

saif_map

Manages the SAIF name-mapping mechanism for reading SAIF files.

```
string saif_map
[-start]
[-end]
[-reset]
[-report]
[-get_name]
[-set_name names]
[-add_name names]
[-remove_name names]
[-clear_name]
[-get_object_names name]
[-create_map]
[-write_map filename]
[-read_map filename]
[-type type]
[-inverted]
[-instances objects]
[-no_hierarchical]
[-columns columns]
[-sort columns]
[-rtl_summary]
[-missing_rtl]
[-input SAIF_file]
[-source_instance SAIF_instance_name]
[-target_instance target_instance_name]
[-review]
[-preview]
[-hsep character]
[object_list]
[-nosplit]
```

save_upf

Writes out the UPF commands in the specified file.

```
collection save_upf
upf_file_name
[-supplemental supf_file_name]
[-include_supply_exceptions]
[-full_upf]
```

set_cell_internal_power

Sets or removes the `power_value` attribute on the specified pins. The value represents the power consumption for a single toggle of each pin.

```
status set_cell_internal_power  
pin_list [power_value [unit]] | -delete_all
```

set_clock_gate_latency

Specifies clock network latency values to be used for clock-gating cells, as a function of clock domain, clock-gating stage, and fanout.

```
status set_clock_gate_latency  
[-clock clock_list]  
[-overwrite]  
-stage cg_stage  
-fanout_latency cg_fanout_list
```

set_clock_gating_objects

Forces the enabling or disabling of clock gating for specified objects in the current design, overriding all conditions necessary for automatic RTL clock gating, by the `compile_ultra -gate_clock` command. Object types can be register, hierarchical cell, power domain or design.

```
status set_clock_gating_objects  
[-force_include object_list]  
[-exclude object_list]  
[-include object_list]  
[-undo object_list]
```

set_clock_gating_registers

Forces the enabling or disabling of clock gating for the specified registers in the current design, overriding all conditions necessary for automatic RTL clock gating, performed by the `compile_ultra -gate_clock` command.

```
status set_clock_gating_registers  
[-include_instances register_list]  
[-exclude_instances register_list]  
[-undo register_list]
```

set_clock_gating_enable

Controls the signals used as clock gating enable in the execution of `compile_ultra -gate_clock` command.

```
status set_clock_gating_enable
[-exclude objects_to_exclude]
[-undo objects_to_remove_exclusion]
```

set_clock_gating_style

Sets the clock-gating style for the clock-gate insertion and replacement.

```
status set_clock_gating_style
[-sequential_cell none | latch]
[-minimum_bitwidth minsize_value]
[-setup setup_value]
[-hold hold_value]
[-positive_edge_logic {cell_list | integrated [active_low_enable]
[invert_gclk]]]
[-negative_edge_logic {cell_list | integrated [active_low_enable]
[invert_gclk]]]
[-control_point none | before | after]
[-control_signal scan_enable | test_mode]
[-observation_point true | false]
[-observation_logic_depth depth_value]
[-max_fanout max_fanout_count]
[-num_stages num_stages_count]
[-no_sharing]
[-instances {instances_list}]
[-power_domains {power_domain_list}]
[-designs {designs_list}]
```

set_cost_priority

Sets the `cost_priority` attribute to a specified value on the current design.

```
status set_cost_priority
[-default]
[-delay]
cost_list
[-design_rules]
[-min_delay]
```

set_design_attributes

Sets the specified attributes and their value on required cells.

```
string set_design_attributes
[-elements list]
[-models list]
[-attribute name_value_pair]*
```

set_dft_clock_gating_configuration

Specifies the clock-gating configuration for a design.

```
status set_dft_clock_gating_configuration
[-exclude_elements object_list]
[-dont_connect_cgs_of cell_list]
```

set_dft_clock_gating_pin

Specifies the test pin of a clock-gating cell in a design. The main purpose of this command is to identify the unconnected test pins of the clock-gating cells that were not inserted by Power Compiler. These pins are connected to test ports when you run the `insert_dft` command.

```
status set_dft_clock_gating_pin
object_list
-pin_name instance_pin_name
[-control_signal ScanEnable | TestMode | scan_enable | test_mode]
[-active_state 1 | 0]
```

set_dft_power_control

Specifies the power controller block instance in a design.

```
status set_dft_power_control power_controller_hierarchical_instance_name
```

set_domain_supply_net

Sets the primary power net and primary ground net of an already existing power domain.

```
status set_domain_supply_net
domain_name
-primary_power_net supply_net_name
-primary_ground_net supply_net_name
```

set_dont_use

Sets the dont_use attribute on library cells to exclude them from the target library during optimization.

```
int set_dont_use [-power] object_list
```

set_equivalent

Specifies power supplies as functionally or electrically equivalent.

```
status set_equivalent
[-function_only]
[-nets supply_net_name_list]
[-sets supply_set_name_list]
```

set_isolation

Defines the UPF isolation strategy for the power domains in the design. Does not apply to bidirectional ports.

```
status set_isolation
isolation_strategy_name
-domain power_domain
[-isolation_power_net isolation_power_net]
[-isolation_ground_net isolation_ground_net]
[-isolation_supply_set isolation_supply_set]
[-clamp_value 0 | 1 | latch]
[-applies_to inputs | outputs | both]
[-source source_supply_set_name]
[-sink sink_supply_set_name]
[-diff_supply_only true | false]
[-elements objects]
[-exclude_elements element_list]
[-no_isolation]
[-force_isolation]
[-name_prefix prefix_string]
[-name_suffix suffix_string]
[-update]
```

set_isolation_cell

Sets the specification of an isolation cell in the library.

```
status set_isolation_cell
cell_name
[-data_pin {data_pin_name}]
[-enable_pin {enable_pin_name}]
```

set_isolation_control

Provides additional options needed for creating isolation cells. This command is needed with most set_isolation commands.

```
status set_isolation_control
isolation_strategy
-domain power_domain
-isolation_signal isolation_signal
[-isolation_sense low | high]
[-location self | parent | fanout]
```

set_leakage_power_model

Specifies the model to be optimized by leakage optimizations.

```
status set_leakage_power_model
[-type model_name]
[-mvth_weights weights]
[-reset]
```

set_level_shifter

Sets a strategy for level shifting during implementation. This command does not apply to bidirectional ports.

```
status set_level_shifter
level_shifter_name
-domain domain_name
[-elements list]
[-exclude_elements element_list]
[-applies_to inputs | outputs | both]
[-threshold value]
[-rule low_to_high | high_to_low | both]
[-location self | parent | automatic]
[-no_shift]
[-force_shift]
[-update]
```

set_level_shifter_cell

Sets on-the-fly specification of a library level-shifter cell.

```

status set_level_shifter_cell
cell_name
[-cell_type cell_type]
[-cell_input_voltage_range {lower_range upper_range}]
[-cell_output_voltage_range {lower_range upper_range}]
[-std_cell_main_rail_pg_pin pg_pin_name]
[-data_pin data_pin_name]
[-input_voltage_range {lower_range upper_range}]
[-output_voltage_range {lower_range upper_range}]
[-input_signal_level signal_level]
[-enable_pin enable_pin_name]
[-enable_signal_level signal_level]
[-output_signal_level signal_level]

```

set_multi_vth_constraint

Sets the maximum percentage of cells, by count or by area, that the design can contain, from the specified low-threshold-voltage groups.

```

status set_multi_vth_constraint
[-lvth_groups groups]
[-lvth_percentage percent_value]
[-cost cost]
[-type type]
[-include_blackboxes]
[-reset]

```

set_port_attributes

Sets the specified attributes and their value on the ports.

```

status set_port_attributes
[-model model_name]
[-ports portlist]
[-elements elementlist]
[-applies_to inputs | outputs | both]
[-attribute atname atvalue]
[-clamp_value 0 | 1 | latch]
[-receiver_supply supply_set_ref]
[-driver_supply supply_set_ref]
[-repeater_supply supply_set_ref]
[-feedthrough]
[-unconnected]

```

set_power_clock_scaling

Sets clock frequency scaling so that the SAIF and synthesis power analysis numbers match. If no *clock_objects* are specified in the command, then only the *-ratio* option is allowed.

```
status set_power_clock_scaling
[-period period_value | -ratio ratio_value]
[clock_objects]
```

set_power_guide

Sets an existing exclusive move bound as a power guide or power well. This power guide is used as an always-on power guide.

```
status set_power_guide
-name exclusive_name
[-guard_band_x horizontal_guard_band_width]
[-guard_band_y vertical_guard_band_width]
```

set_power_prediction

Sets the power prediction mode for `compile_ultra` or `compile_ultra -incremental`. This command is supported only in topographical mode.

```
int set_power_prediction [true | false] [-ct_references lib_cell_list]
```

set_power_switch_cell

Sets the specification of a library power-switch cell.

```
status set_power_switch_cell
cell_name
[-cell_type coarse_grain | fine_grain]
[-is_macro]
[-switch_pin {pin_name}]
[-pg_pin {pin_name switch_function pg_function}]
```

set_query_rules

Defines rules for rule-based query.

```

status set_query_rules
[-hierarchical_separators separator_list]
[-bus_name_notations bus_name_list]
[-class class_list]
[-regsub regsub]
[-regsub_cumulative]
[-wildcard]
[-suffix suffix_list]
[-verbose]
[-nocase]
[-reset]
[-show]

```

set_related_supply_net

Associates a supply net to the port of the design or the pin of a cell.

```

status set_related_supply_net
[-power power_net_name]
[-ground ground_net_name]
[-object_list objects]
[-reset]
[supply_net_name]

```

set_replace_clock_gates

Sets directives for clock gate replacement. Forces the enabling or disabling of clock gate replacement for specified combinational cells in the current design. Also sets the edge type for modules or black-box cells that otherwise could not be replaced. The cells are replaced by executing the `replace_clock_gates` command.

```

int set_replace_clock_gates
[-include_cells cell_list]
[-exclude_cells cell_list]
[-rising_edge_clock pin_list]
[-falling_edge_clock pin_list]
[-undo object_list]

```

set_retention

Defines the UPF retention strategy for the power domains in the design.

```

status set_retention
retention_strategy
-domain power_domain
[-retention_power_net retention_power_net]
[-retention_ground_net retention_ground_net]
[-retention_supply_set retention_supply_set]
[-no_retention]
[-elements objects]
[-save_condition {boolean_function}]
[-restore_condition {boolean_function}]
[-retention_condition {boolean_function}]
[-update]

```

set_retention_cell

Sets the specification of a library retention cell.

```

Boolean set_retention_cell
cell_name
[-cell_type retention_type]
[-retention_pin {pin_name pin_type disable_value}]

```

set_retention_control

Defines the UPF retention control signals for the defined UPF retention strategy.

```

status set_retention_control
retention_strategy
-domain power_domain
-save_signal {save_signal save_sense}
-restore_signal {restore_signal restore_sense}
[-assert_r_mutex {net_name sense}]
[-assert_s_mutex {net_name sense}]
[-assert_rs_mutex [{net_name sense}]

```

set_retention_control_pins

Converts the retention register library cell attributes in the old library format to the ones that can be used in the \$retain flow. The \$retain flow requires retention register library cells to have new retention cell attributes, which is different from the original power-gating flow.

```

status set_retention_control_pins
[-type style]
[-power_pin_index power_pin | -library_pin library_pin_name]
[-is_save_pin | -is_restore_pin | -is_save_restore_pin]
lib_or_lib_cell_list

```

set_retention_elements

Specifies a list of elements that can be used later with the `set_retention` command. The specified list is tied to the scope where the `set_retention_elements` command is defined.

```
status set_retention_elements retention_list_name
-elements element_list
```

set_scenario_options

Sets the scenario options for one or more scenarios.

```
status set_scenario_options
[-scenarios scenario_list]
[-setup true | false]
[-hold true | false]
[-leakage_power true | false]
[-dynamic_power true | false]
[-cts_mode true | false]
[-cts_corner min | max | min_max | none]
[-reset_all true | false]
```

set_scope

Specifies the current scope.

```
string set_scope [instance]
```

set_self_gating_objects

Forces the enabling or disabling of self-gating for specified objects in the current design, overriding all conditions necessary for automatic self-gating, by the `compile_ultra -self_gating` command. Objects types can be register, hierarchical cell, power domain, or design.

```
status set_self_gating_objects
[-force_include object_list]
[-exclude object_list]
[-include object_list]
[-undo object_list]
```

set_self_gating_options

Sets the self-gating options for the self-gate insertion. This command is supported only in topographical mode.


```
string set_self_gating_options
[-min_bitwidth min_bitwidth_count]
[-max_bitwidth max_bitwidth_count]
[-interaction_with_clock_gating none | insert | merge]
```

set_switching_activity

Sets the switching activity annotation on nets, pins, ports, and cells in the current design.

```
status set_switching_activity
[-static_probability static_probability]
[-toggle_rate toggle_rate]
[-state_condition state_condition]
[-path_sources path_sources]
[-rise_ratio rise_ratio]
[-period period_value | -base_clock clock]
[-type object_type_list]
[-hierarchy]
[object_list]
[-verbose]
```

set_switching_activity_profile

Generates switching activity profiles and sets the profiles on the input buses ports in the current design.

```
status set_switching_activity_profile
[-uniform {static_probability toggle_rate}]
[-linear {{breakpoint_0 static_probability_0 toggle_rate_0}
{breakpoint_1 static_probability_1 toggle_rate_1}}]
[-normal_dist {std_dev temp_corr sample_rate is_signed}]
[-period period_value | -base_clock clock]
[object_list]
[-all_buses]
[-verbose]
```

set_upf_query_options

Enables or disables bus and structure querying support for the `-elements` option for the `set_retention`, `set_isolation`, and `map_retention_cell` commands.

```
status set_upf_query_options
-bus_struct_mode true | false
```

unset_power_guide

Unsets an existing power guide to be similar to an exclusive move bound.

```
unset_power_guide
[power_guide_list]
```

upf_version

Displays the version of UPF currently used to interpret the UPF commands

```
string upf_version
[version]
```

write_saif

Writes a backward Switching Activity Interchange Format (SAIF) file.

```
status write_saif
-output file_name
[-instances instances]
[-no_hierarchy]
[-rtl]
[-propagated]
[-exclude_sdpd]
```

write_script

Writes shell commands to save the current settings.

```
int write_script
[-hierarchy]
[-no_annotated_check]
[-no_annotated_delay]
[-no_cg]
[-full_path_lib_names]
[-nosplit]
[-format dctcl | dcsh]
[-include loop_breaking]
[-output file_name]
```

Power Compiler Variables

Power Compiler defines a set of variables that are used to control its behavior.

abstraction_enable_power_calculation

Performs power calculations on a design that is to be used as a block abstraction.

The default for this variable is `true`.

`compile_power_domain_boundary_optimization`

Disables boundary optimization across power domain boundaries when set to `false`.

The default for this variable is `true`.

`enable_golden_upf`

Enables the golden UPF mode when set to `true`.

The default for this variable is `false`.

`enable_rule_based_query`

Enables or disables rule-based matching.

The default for this variable is `false`.

`golden_upf_report_missing_objects`

Enables reporting of missing objects during Golden UPF reapplication.

The default for this variable is `false`.

`hdlin_enable_upf_compatible_naming`

Controls HDL Compiler naming style settings to make it easier to apply the same UPF file across multiple tools at the RTL level.

The default for this variable is `false`.

`link_allow_upf_design_mismatch`

This variable controls the dirty data handling features, which include the `mv_no_main_power_violations` and `mv_use_std_cell_for_isolation` variables and link library cells with PVT violations.

The default for this variable is `true`.

mv_allow_ls_on_leaf_pin_boundary

Allows level-shifter insertion on leaf pin (such as macro cell pin) boundaries.

The default for this variable is `false`.

mv_allow_va_beyond_core_area

Allows voltage area to be created even if the voltage area is not fully enclosed inside the core area.

The default for this variable is `false`.

mv_input_enforce_simple_names

Enforces the use of simple names for restricted commands according to the IEEE 1801 (UPF) standard.

The default for this variable is `false`.

mv_insert_level_shifters_on_ideal_nets

Directs automatic level-shifter insertion to insert level shifters on ideal nets.

The default for this variable is `""`.

mv_make_primary_supply_available_for_always_on

Gives preference to load and driver supplies when set to `false`. This might result in seeing more always-on buffers in the netlist.

The default for this variable is `true`, which results in regular buffer insertion on feedthrough nets when the primary power supply is used.

mv_no_always_on_buffer_for_redundant_isolation

Allows normal buffers to be used at nets driving the data input of redundant isolation cells.

The default for this variable is `false`.

mv_no_cells_at_default_va

Controls whether the tool can place new buffers at the default voltage area.

The default for this variable is `false`.

mv_no_main_power_violations

Selects Standard Cell Main Rail (SCMR) as the main power supply for level shifters.

The default for this variable is `true`.

mv_output_enforce_simple_names

Enforces the use of simple names for restricted commands as per the IEEE 1801 (UPF) standard.

The default for this variable is `false`.

mv_output_upf_line_indent

Sets the amount of indentation spaces added to lines when the `save_upf` command is splitting long commands onto multiple lines.

The default for this variable is `2`.

mv_output_upf_line_width

Controls whether the `save_upf` command outputs long commands onto multiple lines. Also sets the threshold for splitting lines.

The default for this variable is `0`.

mv_skip_opcond_checking_for_unloaded_level_shifter

Skips operating condition checking for level shifters with unconnected output pins.

The default for this variable is `false`.

mv_upf_tracking

Controls whether the UPF tracking feature is enabled in the current session.

The default for this variable is `true`.

mv_use_std_cell_for_isolation

Allows the tool to insert standard cells as isolation cells.

The default for this variable is `false`.

physopt_power_critical_range

Specifies a margin of slack for cells during leakage power optimization. If a cell has a slack less than the power critical range, power optimization is not done for the cell.

The default for this variable is `-1.04858e+06`.

power_cg_all_registers

Specifies to the `insert_clock_gating` command whether to clock gate all registers, including those that do not meet the necessary requirements.

The default for this variable is `false`.

power_cg_auto_identify

Activates automatic identification of Power Compiler inserted clock-gating circuitry from a structural netlist.

The default for this variable is `false`.

power_cg_balance_stages

Controls whether gate stage balancing is on or off when you run the `compile [-incremental_mapping] -gate_clock` or `compile_ultra [-incremental] -gate_clock` command.

The default for this variable is `false`.

power_cg_cell_naming_style

Specifies the naming style for clock-gating cells created by the `insert_clock_gating` command.

The default for this variable is `""`.

power_cg_derive_related_clock

Derives the clock domain relationship between registers from the hierarchical context.

The default for this variable is `false`.

power_cg_designware

Performs clock gating on DesignWare sequential components in the design.

The `power_cg_designware` variable will be obsolete in a future release. Clock gating insertion with `compile_ultra -gate_clock` automatically inserts clock gates in DesignWare modules.

The default for this variable is `false`.

power_cg_enable_alternative_algorithm

Controls whether the `insert_clock_gating`, `compile -gate_clock`, and `compile_ultra -gate_clock` commands use an alternative algorithm to find registers to gate.

The default for this variable is `false`.

power_cg_ext_feedback_loop

Controls whether external feedback loops should be used to generate the enable condition for a register with its enable pin tied to logic 1.

The default for this variable is `true`.

power_cg_flatten

Specifies to different ungroup commands whether or not to flatten Synopsys clock-gating cells.

The default for this variable is `false`.

power_cg_gated_clock_net_naming_style

Specifies the naming style for gated clock nets created by the `insert_clock_gating` command.

The default for this variable is `""`.

power_cg_ignore_setup_condition

Ignores the setup condition for latch-free clock gating.

The default for this variable is `false`.

power_cg_inherit_timing_exceptions

Controls whether the `compile -gate_clock` and `compile_ultra -incremental -gate_clock` commands automatically infer the timing exceptions defined on registers onto the enable pin of the clock gate that is gating these registers.

The default for this variable is `false`.

power_cg_iscgs_enable

Controls whether the `set_clock_gating_style`, `remove_clock_gating_style`, `compile`, and `compile_ultra` commands use the instance-specific clock-gating style.

The default for this variable is `false`.

power_cg_module_naming_style

Specifies the naming style for clock gating modules created by the `insert_clock_gating` command.

The default for this variable is `""`.

power_cg_physically_aware_cg

Enables a clock-gating insertion and optimization flow that considers the physical information.

The default for this variable is `false`.

power_cg_print_enable_conditions

Reports the enable conditions of registers and clock gates during clock-gate insertion.

The default for this variable is `false`.

power_cg_print_enable_conditions_max_terms

Specifies the maximum number of product terms to be reported in the sum of product expansion of the enable condition.

The default for this variable is 10.

power_cg_reconfig_stages

Controls the reconfiguration of multistage clock gates during the compile
[-incremental_mapping] -gate_clock and compile_ultra [-incremental]
-gate_clock commands.

The default for this variable is `false`.

power_cg_sequential_clock_gating

Enables the execution of sequential clock gating.

The default for this variable is `false`.

power_default_static_probability

Specifies the default static probability value.

The default for this variable is 0.5.

power_default_toggle_rate

Specifies the default toggle rate value.

The default for this variable is 0.1.

power_default_toggle_rate_type

Specifies the default toggle rate type.

The default for this variable is `fastest_clock`.

power_do_not_size_icg_cells

Controls whether compile does not size the integrated clock-gating cells in a design to correct DRC violations because doing so might result in lower area and power.

The default for this variable is `true`.

power_enable_clock_scaling

Enables clock frequency scaling when set to `true`.

The default for this variable is `false`.

power_enable_datapath_gating

Enables or disables datapath gating which is a dynamic power optimization technique for a design.

The default for this variable is `false`.

power_enable_one_pass_power_gating

Enables one-pass flow power gating. This variable is for use only in non-UPF mode.

The default for this variable is `false`.

power_enable_power_gating

Enables the power-gating flow that allows the selected retention registers from the target library to be used to map sequential elements. This variable can be used only in non-UPF mode. In UPF mode, use UPF commands to enable the power-gating flow.

The default for this variable is `false`.

power_fix_sdpd_annotation

Specifies whether user-annotated state-dependent or path-dependent switching activity annotation is corrected before it is used.

The default for this variable is `true`.

power_fix_sdpd_annotation_verbose

Specifies whether verbose messages are reported during the fixing of user-annotated state-dependent or path-dependent switching activity.

The default for this variable is `false`.

power_hdlc_do_not_split_cg_cells

Specifies that the `insert_clock_gating` command will not split clock-gating cells to limit their fanout.

The default for this variable is `false`.

power_keep_license_after_power_commands

Affects the amount of time a Power Compiler license is checked out during a shell session.

The default for this variable is `false`.

power_lib2saif_rise_fall_pd

Specifies whether the `lib2saif` command generates forward SAIF files with directives to generate rise and fall dependent and path-dependent toggle counts.

The default for this variable is `false`.

power_low_power_placement

This variable controls the power-aware placement during the `compile_ultra` command in the Synopsys physical guidance flow.

The default for this variable is `false`.

power_min_internal_power_threshold

Specifies the minimum cell internal power value that can be used in power calculations.

The default for this variable is "".

power_model_preference

Specifies the preference between the CCS power and the NLPM models in library cells that have power specified in both models.

The default for this variable is `nlpm`.

power_opto_extra_high_dynamic_power_effort

Instructs the `compile` and `compile_ultra` commands to invoke more dynamic power optimization algorithms.

The default for this variable is `false`.

power_preserve_rtl_hier_names

Preserves the hierarchy information of the RTL objects in the RTL design.

The default for this variable is `false`.

power_rclock_inputs_use_clocks_fanout

Specifies whether clock network objects in an input port fanout are used to infer the input port's related clock.

The default for this variable is `true`.

power_rclock_unrelated_use_fastest

Specifies whether the fastest clock is set as the related clock of a design object when a related clock is not inferred by the related clock inference mechanism.

The default for this variable is `true`.

power_rclock_use_async_inputs

Specifies whether the inferred related clock on an asynchronous pin of a flip-flop is used to determine the inferred related clock on the cell's outputs.

The default for this variable is `false`.

power_remove_redundant_clock_gates

Specifies to the `compile -incremental` and `physopt -incremental` commands to remove redundant clock-gating cells.

The default for this variable is `true`.

power_rtl_saif_file

Defines for the `rtl2saif` command where to store the forward-annotation SAIF file, if you do not specify the `-output` option.

The default for this variable is `power_rtl.saif`.

power_sa_propagation_verbose

Specifies the default verbose mode used when propagating switching activity.

The default for this variable is `false`.

power_same_switching_activity_on_connected_objects

Forces the tool to use the last user-annotated switching activity data on all connected tool objects.

The default for this variable is `false`.

power_sdpd_message_tolerance

Specifies the tolerance value for issuing warning and information messages during fixing of user-annotated state-dependent and path-dependent switching activity.

The default for this variable is `1e-05`.

synlib_enable_analyze_dw_power

Records the power information of ungrouped DesignWare designs.

The default for this variable is 0.

upf_allow_DD_primary_with_supply_sets

Controls the use of domain-dependent supply nets in the design as the primary supply of the power domain when supply sets are used in the design.

The default for this variable is false.

upf_allow_refer_before_define

Allows UPF commands to refer to block-level UPF objects that are not defined in the top-only UPF. Set this variable to true to allow references to objects not defined in the top-only UPF.

The default is false.

upf_auto_iso_clamp_value

Sets the clamp value for the inferred isolation strategy.

The default for this variable is 0.

upf_auto_iso_enable_source

Sets the isolation signal for the inferred isolation strategy.

The default for this variable is root_cell.

upf_auto_iso_isolation_sense

Sets the isolation sense for the inferred isolation strategy.

The default for this variable is low.

upf_block_partition

Allows the subsequent `save_upf` command to skip saving the UPF data for the specified blocks within a top-level design. A list of blocks follows this variable. This variable is tied to the current design.

There is no default for this variable.

upf_charz_allow_port_punch

Allows UPF-related port punching to occur within the `characterize` command on the blocks being characterized.

The default for this variable is `true`.

upf_charz_enable_supply_port_punching

Allows UPF-related supply port punching to occur within the `characterize` command on the blocks being characterized.

The default for this variable is `true`.

upf_charz_max_srsn_messages

Sets the maximum number of error and warning messages from the `set_related_supply_net` command that can be printed when characterizing a block.

The default for this variable is 10.

upf_create_implicit_supply_sets

Allows supply handles of the power domains to be created.

The default for this variable is `false`.

upf_enable_legacy_block

Allows UPF 1.0-style design blocks to be integrated into a UPF 2.0-style design.

The default for this variable is `true`.

upf_enable_relaxed_charz

Allows flexible partitioning of power domains and supports -location parent for isolation cells in a hierarchical flow.

The default for this variable is `true`.

upf_extension

Disables the writing of UPF extension commands in the `save_upf` command.

The default for this variable is `true`.

upf_isols_allow_instances_in_elements

Controls the specification of instances with the `-elements` option of the `set_isolation` command.

The default for this variable is `true`.

upf_iso_filter_elements_with_applies_to

Controls the filtering of design elements specified with the `-elements` option and the `-applies_to` option of the `set_isolation` command.

The default for this variable is `ENABLE`.

upf_levshi_on_constraint_only

Inserts level shifters only at the domain boundaries with a level shifter strategy.

The default for this variable is `false`.

upf_name_map

Specifies the name map files used during the reapplication of golden UPF file to designs.

The default for this variable is `""`.

upf_report_isolation_matching

Allows reporting of isolation strategies matching (UPF-073).

The default for this variable is `false`.

upf_skip_ao_check_for_els_input

Directs automatic level-shifter insertion to ignore always-on checking when choosing the input supplies for an enabled level shifter.

The default for this variable is `true`.

upf_suppress_etm_model_checking

Controls ETM model checking for a UPF file. Set to `true` to disable ETM model checking in the UPF for a macro cell.

The default for this variable is `false`.

upf_suppress_message_in_black_box

Suppresses the messages due to missing objects when UPF commands are loaded at the black box scope.

The default for this variable is `true`.

upf_suppress_message_in_etm

Suppresses the messages for commands that are not allowed on ETM scope.

The default for this variable is `true`.

